

ARBEITSBERICHTE DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG (INFORMATIK)

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

HERAUSGEBER: H. Billing, W. Händler, U. Herzog,
F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees, H. Niemann,
D. Seitzer, B. Schmidt, H.J. Schneider, H. Wedekind

TRANSAKTIONEN ALS GRUNDLAGE ZUR STRUKTURIERUNG UND INTEGRITÄTSSICHERUNG IN DATENBANK-ANWENDUNGSSYSTEMEN

Dieter Steinbauer

DISSERTATION

BAND 16 · NUMMER 13 · ERLANGEN · DEZEMBER 1983

1984 # 100



ISSN 0344 - 3515

TRANSAKTIONEN ALS GRUNDLAGE ZUR STRUKTURIERUNG UND INTEGRITÄTSSICHERUNG IN DATENBANK-ANWENDUNGSSYSTEMEN

Der Technischen Fakultät
der Universität Erlangen-Nürnberg

zur Erlangung des Grades

DOKTOR - INGENIEUR

vorgelegt

von

Dieter Steinbauer

Erlangen 1983

TRANSAKTIONEN ALS GRUNDLAGE
ZUR STRUKTURIERUNG UND INTEGRITÄTSSICHERUNG
IN DATENBANK-ANWENDUNGSSYSTEMEN

Als Dissertation genehmigt von der
Technischen Fakultät der
Universität Erlangen-Nürnberg

Tag der Einreichung: 28.10.1983

Tag der Promotion: 20.12.1983

Dekan: Prof. Dipl.-Ing. G. Vetter

Berichterstatter: Prof. Dr. H. Wedekind

Priv.-Doz. Dr. S. Keramidis

Vorwort

Das Transaktionsprinzip entwickelte sich zum Paradigma der Bearbeitung von Datenbanken. Es wurde von Eswaran, Gray, Lorie und Traiger <EGLT76> im Rahmen der Überlegungen zum System R entwickelt.

Die in dieser Arbeit vorgenommene Untersuchung der Struktur von Transaktionen anhand von Anforderungen aus der Anwendung ergab eine Klassifikation in schematische, reglementierte und freie Transaktionen, die sich auf die Software-Architektur eines Datenbank-Anwendungssystems auswirkt. Die Erweiterung der Externen Schemata der Datenbank zu Virtuellen Abstrakten Datentypen ergab sich als eine Konsequenz für den Aufbau eines Anwendungssystems. Die vorgenommene Differenzierung der Transaktionen eröffnete zusätzliche Möglichkeiten zur Kontrolle und zur Sicherung der Integritätsanforderungen in einem Datenbank-Anwendungssystem.

Im Ergebnis kann man Jim Gray (<Gra81>, S.153) nur zustimmen, wenn er schreibt:

"I am optimistic that the transaction concept provides a convenient abstraction for structuring applications."

Herrn Prof. Dr. H. Wedekind möchte ich für die vielfältigen Anregungen, die intensive wissenschaftliche Betreuung und die sonstige gewährte Förderung aufrichtig danken.

Herrn Dr.-Ing. habil. S. Keramidis danke ich für sein in zahlreichen Gesprächen gezeigtes Interesse und für die Übernahme des Zweitgutachtens.

Außerdem bedanke ich mich bei meinen Kollegen Werner Eberlein und Theo Müller für die in kritischen Diskussionen gegebenen Anstöße.

Frau B. Knechtel danke ich insbesondere für die gewährte Unterstützung beim Anfertigen der Abbildungen.

Meine Frau Gabi verdient liebevollen Dank nicht nur für ihr stets gezeigtes Verständnis, sondern auch für die Einführung in die Welt des pharmazeutischen Außendienstes, wodurch mir die Darstellung eines realitätsnahen Anwendungsbeispiels erleichtert wurde.

Inhaltsverzeichnis

0. Einleitung	1
1. Einführende Bemerkungen und Grundlagen	3
1.1. Die statische Komponente eines Datenbank- Anwendungssystems	4
1.1.1. Die Datenbank	5
1.1.2. Sichten der Datenbank	10
1.2. Ansätze zu einer operationalen Komponente eines Datenbank-Anwendungssystems	13
1.2.1. Mengenorientierte Datenbank-Änderungs- operationen	15
1.2.2. Das Konzept der Transaktion	18
1.2.3. Ansätze zum Transaktionsentwurf und zum Aufbau eines Transaktionssystems	23
1.3. Integritätskontrolle - Wechselwirkungen zwischen Transaktionen und Datenbank	26
2. Rekonstruktion und Struktur von Transaktionen	34
2.1. Rekonstruktion der Transaktionsstrukturen aus den Handlungsstrukturen	35
2.1.1. Erste Bemerkungen zum Begriff Handlung	36
2.1.2. Handlungsmuster oder -schema, ...	39
2.1.3. Begründung und Rechtfertigung von Handlungen ..	48
2.1.4. Die Gray'schen Kriterien	51
2.2. Die Struktur von Transaktionen	53
2.2.1. Ein bekannter Ansatz - Die Transaktions- prozedur	54
2.2.2. Von universellen Datenbank-Operationen zu anwendungsspezifischen Operationen	63
2.2.3. Die Transaktion als Prozeß	68
3. Zum strukturellen Aufbau von Datenbank-Anwendungs- systemen	74
3.1. Das Externe Schema als Basis für Aktionen	76
3.2. Aufbau einer operationalen Sicht der Datenbank ...	94

3.3. Transaktionsorientierte Strukturierung eines Datenbank-Anwendungssystems	106
3.4. Mögliche Konsequenzen für die Datenbank-Architektur	113
4. Die Schematisierbarkeit der Transaktionen - Basis eines Transaktionssystem	120
4.1. Klassifikation der Transaktionen nach ihrer Schematisierbarkeit	121
4.2. Schematische Transaktionen	126
4.2.1. Rekonstruktion	127
4.2.2. Strukturelle Darstellung	128
4.3. Reglementierte Transaktionen	135
4.3.1. Rekonstruktion	136
4.3.2. Strukturelle Darstellung	139
4.3.3. Möglichkeiten zur Implementierung	147
4.4. Freie Transaktionen	152
4.4.1. Rekonstruktion	153
4.4.2. Strukturelle Darstellung und Implementierung ..	154
4.5. Zusammenfassung der Aspekte zur Schematisierung von Transaktionen	157
5. Integritätskontrolle in Datenbank-Anwendungssystemen.....	159
5.1. Rekonstruktion der Integrität	160
5.1.1. Semantische, strukturelle und technische Integrität aus der Sicht der Anwendung	161
5.1.2. Eine Erweiterung der Integrität für Datenbank-Anwendungen	171
5.1.2.1. Finale und afinale Normen - der intentionale Aspekt	173
5.1.2.2. Das Handeln nach Normen - der deontische Aspekt	175
5.2. Einige Prinzipien der Integritätskontrolle	180
5.3. Integritätskontrolle auf der Datenbank und in den Sichten	190
5.3.1. Strukturelle Datenintegrität	191

5.3.2. Semantische Datenintegrität	200
5.3.3. Finale deontische Integrität	206
5.3.4. Globale und lokale Integrität - Beziehungen zwischen Datenbank und Sichten	215
5.4. Integritätskontrolle auf der Basis der Datenbank-Anwendungsmoduln	219
5.4.1. Möglichkeiten zur Kontrolle der Dateninte- grität	220
5.4.2. Möglichkeiten zur Kontrolle operationaler Integrität	227
5.5. Transaktionsorientierte Integritätskontrolle	233
5.5.1. Allgemeine Prinzipien transaktionsorientierter Integritätskontrolle	234
5.5.2. Unterschiede in der Integritätskontrolle bei den verschiedenen Transaktionsarten	239
5.5.2.1. Integritätskontrolle bei schematischen Transaktionen	240
5.5.3.2. Integritätskontrolle bei reglementierten Transaktionen	244
5.5.4.3. Integritätskontrolle bei freien Trans- aktionen	249
5.6. Zusammenfassende Bemerkungen zur Integritäts- kontrolle	252
6. Eine subjektbezogene Erweiterung der Integritäts- kontrolle	254
6.1. Gründe und Struktur einer subjektbezogenen Erweiterung der Integritätskontrolle	255
6.2. Subjektbezogene deontische Integritätsbedingungen	260
6.4. Subjektbezogene Integrität als ein Beitrag zur Sicherheitskontrolle	264
7. Zusammenfassung	266
8. Literatur	268

Anhang 1.

Beispiel: Datenbank - Pharmaaußendienst

Anhang 2.

Beispiel: Statische und Operationale Sichten
für einen Pharmaaußendienstmitarbeiter

Anhang 3.

Beispiel: Einige typische Transaktionen
für einen Pharmaaußendienstmitarbeiter

Anhang 4.

Beispiel: Einige typische Transaktionen
für die Pharmaaußendienstleitung

Anhang 5.

Überblick: Datenbank-Anwendungssystem
für den Pharmaaußendienst

0. Einleitung

Die Idee der Transaktion ist eigentlich Tausende von Jahren alt, meint Jim Gray (<Gra81>, S.153). Ihm und seinen Kollegen <EGLT76> kommt jedoch das Verdienst zu, dieses Prinzip für die Bearbeitung von Datenbanken wiederentdeckt zu haben.

Das Transaktionsprinzip entwickelte sich zum Paradigma der Bearbeitung von Datenbanken. Jim Gray <Gra81> räumt aber ein, daß das zur Zeit vorhandene Konzept und dessen Realisierung nicht für alle Datenbank-Anwendungen ausreichend ist.

In dieser Arbeit wird aufbauend auf der Kritik am Transaktionskonzept erörtert, wie ein Datenbank-Anwendungssystem transaktionsorientiert strukturiert werden kann und welche Konsequenzen sich hierbei auf die Erhaltung der Integrität der Datenbank und des Anwendungssystems ergeben.

Das Transaktionskonzept wird untersucht, inwieweit es sich generell als Grundlage zur Automatisierung von Handlungen einer Anwendung eignet (Kapitel 2). Das Konzept wird dabei zur allgemeinen Handlungstheorie in Beziehung gesetzt.

Aufbauend auf dem Transaktionskonzept wird eine Architektur für Datenbank-Anwendungssysteme entwickelt, die der Forderung nach Einbeziehung des Konzepts der Abstrakten Datentypen (<Cod82>, S.116) nachkommt. Dies geschieht im strukturellen Aufbau der Datenbank-Anwendungssysteme durch die Bereitstellung einer operationalen Sicht auf die Datenbank. Diese operationale Sicht wird durch eine Menge "Virtueller" Abstrakter Datentypen realisiert, die ihre Objekte über die Datenbank verwalten. Auf dieser operationalen Sicht können dann die Transaktionen realisiert werden (Kapitel 3).



Die Transaktionen werden in ihrem strukturellen Aufbau differenziert in schematische, reglementierte und freie Transaktionen (Kapitel 4), wodurch sich Unterschiede in der Struktur der Spezifikation und Implementation ergeben. Diese Klassifikation erweitert die Möglichkeiten des Transaktionskonzepts im Hinblick auf die Kontrolle der Integrität.

Die Kontrolle der Integrität ist ein zentraler Punkt in der Bearbeitung einer Datenbank. Aufgrund der untersuchten Struktur eines Anwendungssystems und der Struktur der Transaktionen können die Möglichkeiten zur Integritätskontrolle weiter ausgebaut werden. Die bekannten Arten der Integritätskontrolle in Datenbank-Anwendungssystem wird durch die Einführung einer deontischen Integrität auf der Basis der deontischen Logik ergänzt (Kapitel 5).

Die deontische Integrität, die die Zulässigkeit von Transaktionen behandelt, erlaubt, wenn sie subjektbezogen betrachtet wird, eine "Inputkontrolle" gegen mißbräuchliche Veränderungen der Datenbank. Sie ist neben der "Outputkontrolle" <Schl82> somit auch ein Beitrag zur Sicherheit der Datenbank (Kapitel 6).

1. Einführende Bemerkungen und Grundlagen

Um die Strukturierung und den Aufbau einer operationalen Komponente eines Datenbank-Anwendungssystems als ein System von Transaktionen behandeln zu können, sind zuerst einige Grundbegriffe zu skizzieren, die die Basis der weiteren dargelegten Überlegungen bilden. Ziel ist es dabei nicht, einen vollständigen Überblick über den Aufbau und die Möglichkeiten existierender Datenbanksysteme zu geben. Vielmehr erfolgt eine Beschränkung auf einige spezifische Punkte, die im unmittelbarem Zusammenhang mit der Erstellung von Datenbank-Anwendungssystemen stehen.

Die hier anstehenden Probleme liegen in der Bereitstellung eines geeigneten Konzepts zur Formulierung und zur Kontrolle transaktionsorientierter Operationen auf der Datenbank. Für die weiteren Betrachtungen wird als Basis eine relationale Datenbank vorausgesetzt. Das Relationale Datenmodell ist durch seinen einfachen Strukturtyp, die Daten in der Form von Relationen darzustellen, und durch die mengenorientierte operationale Schnittstelle anderen herkömmlichen Datenmodellen im Abstraktionsniveau überlegen. Es existieren andererseits genügend Realisierungen relationaler Datenbanksysteme (vgl. <SB83>) im Vergleich zu den sogenannten "höheren" "semantischen" Datenmodellen (<BZ81>, <Che76>, <McL78>).

Auf eine ausführliche Beschreibung des Datenmodells wird verzichtet, da hierfür genügend Standardliteratur zur Verfügung steht (<Ull80>, <Dat81>, <Wed81>). Im weiteren wird speziell auf die formale Beschreibung des Relationalen Datenmodells nach Codd (<Cod70>, <Cod79>) und (<Dat82>, <Pir82>) Bezug genommen. Die Kenntnis der Datenbanksprachen SQL des System R (<Cha76>) bzw. QUEL des Systems INGRES (<RT82>, <Sto76>) wird ebenfalls vorausgesetzt.

1.1. Die statische Komponente eines Anwendungssystems

Im Mittelpunkt des Aufbaus eines Datenbank-Anwendungssystems steht die Datenbank, die auch als die statische Komponente eines Anwendungssystems bezeichnet wird (<FR78>, S.343). Statisch nennt man die Datenbank innerhalb eines Anwendungssystems deshalb, weil die in der Datenbank abgelegten Daten die Zustände der zugrundegelegten Anwendung repräsentieren. Die Form, in der die Daten zu strukturieren sind, ist durch das Relationale Datenmodell bestimmt. Apriori nicht festgelegt ist dagegen, welche Objekte der Miniwelt in welchen Eigenschaften durch eine Relationen zu repräsentieren sind. Die Beschreibung der konkreten Datenstrukturen einer spezifischen Anwendung erfolgt deshalb in einem Konzeptionellen (Datenbank-)Schema.

Ist eine Anwendung komplex, indem sie sich z.B. über mehrere Teilbereiche einer Unternehmung erstreckt, umfaßt die Datenbank z.B. das Rechnungswesen, den Vertrieb, die Fertigung etc., so besitzt jeder dieser Teilbereiche eine lokale, für ihn spezifische Sicht auf die Datenbank. Jeder dieser Teilbereiche greift mit verschiedenen anwendungsspezifischen Operationen auf die Datenbank zu. Aufgabe eines Datenbanksystems ist es auch, diese lokalen Sichten mit ihren Operationen bereitzustellen. Die Beschreibung einer lokalen Sicht wird als Externes Schema bezeichnet (vgl. <Jef76>, <Yor76>).

Als Basis für die operationale Komponente einer Anwendung sind somit folgende (vgl. <Pir82>, S.34) Aspekte eines Datenbanksystems von Interesse:

die Datenbank

- der Zustand der Datenbank

(als die Zustandsbeschreibung einer Anwendung)

- das Konzeptionelle Schema
(als die Präzisierung der Strukturen bzw. Grundbegriffe, in der eine Anwendung beschrieben wird)

die Sichten der Datenbank

- die Externen Schemata
(als die Strukturbeschreibungen der lokalen Sichten auf die Datenbank, abgeleitet aus dem Konzeptionellen Schema)
- der Wert der Sichten
(als eine aus der Datenbank abgeleitete Zustandsbeschreibung eines Teils der Anwendung)

1.1.1. Die Datenbank

Der Zustand einer Datenbank kann aufgefaßt werden als Darstellung der relevanten Sachverhalte oder Situationen einer konkreten Anwendung. "Relevant" bezeichnet dabei genau jene Sachverhalte, die für die in der Anwendung verfolgten Ziel- und Zwecksetzungen von unmittelbarer Bedeutung sind. Für die Benutzer besteht somit die Datenbank aus einer strukturierten Sammlung von Daten, die nichts anderes sind, als Aussagen über Objekte und deren Eigenschaften seiner Anwendung. Der Datenbank liegt damit eine (anwendungsbezogene) Semantik zugrunde, nämlich die Objekte der Anwendung mit ihren Eigenschaften (im weiteren auch als "Miniwelt" bezeichnet). Diese Art der Semantik ist abzugrenzen von der (maschinenbezogenen) Semantik der Repräsentation der Daten im Rechner. Der konkrete syntaktische Aufbau der in der Datenbank abgelegten Daten ist durch das gewählte Datenmodell bestimmt.

Das Datenmodell muß es gestatten, die spezifischen (Begriffs-)Strukturen einer Anwendung in adäquater Weise darzustellen. Im Relationalen Datenmodell geschieht dies auf

der Basis von Relationen. Die einzelnen Tupel­einträge reprä­sentieren die Sachverhalte einer Anwendung, die Bedeutung erhalten sie über die Struktur der Relationen, die für die Grundbegriffe der Anwendung stehen (vgl. <Wed81>, S.176ff).

Eine wesentliche Aufgabe ist es somit, die Grundbegriffe einer Anwendung zu erkennen, zu präzisieren und in entsprechende Datenstrukturen der Datenbank umzusetzen - im Relationalen Modell durch eine oder mehrere Relationen. Eine Präzisierung des Begriffs des Datenbankzustands, soweit er für das Ziel dieser Arbeit nötig erscheint, erfordert zuerst eine Bestimmung der Begriffe Relation und Relationenschema, wie sie im Relationalen Datenmodell verwendet werden (vgl. auch <Dat82>, <Pir82>).

Relation (im Sinne des Relationalen Modells nach Cod79)

Sei $A = \{ A_1, A_2, \dots, A_n \}$ mit $i \in N$ eine Menge von Mengennamen (auch als Attribute bezeichnet) und $ATTRIBUTE-i$ die zugehörigen Mengen, so heißt $\underline{R} \subseteq ATTRIBUTE-1 \times \dots \times ATTRIBUTE-n$ eine n -stellige Relation über A .

Relationenschema

Ist $\underline{R} \subseteq ATTRIBUTE-1 \times \dots \times ATTRIBUTE-n$ eine Relation über $A = \{ A_1, A_2, \dots, A_n \}$, dann heißt $R (A_1, \dots, A_n)$ Relationenschema von $\underline{R}$ mit dem Relationennamen R .

Die Menge $RELATION := ATTRIBUTE-1 \times \dots \times ATTRIBUTE-n$ heißt Definitionsbereich des Relationenschemas (oder kurz der Relation) R .

Datenbankzustand

Der Zustand "db" einer Datenbank "DB" ist durch ein Tupel $(R_1, \dots, R_m)$ von Relationen festgelegt, m steht dabei für die Anzahl der in der Datenbank enthaltenen Relationen.

Der Zustandsraum (Domäne) "DATABASE" der Datenbank ist bestimmt durch $(\text{RELATION-1}, \dots, \text{RELATION-m})$.

Im Konzeptionellen Schema einer Relationalen Datenbank erfolgt die Festlegung der benötigten Begriffsstrukturen einer Miniwelt. Es werden die Relationen einer Datenbank sowie ihre Beziehungen zueinander beschrieben. Die Abhängigkeiten zwischen den Attributen und der Bezug zur Miniwelt wird fixiert, wodurch letztlich eine Festlegung der Begriffstrukturen erreicht wird.

Das Konzeptionelle Schema einer Datenbank besteht aus der Beschreibung der Datenstrukturen (im Relationalen Modell durch Relationenschemata) und einer Menge von Integritätsbedingungen, die alle möglichen Zustände (Relationen über den Definitionsbereich der Relationenschemata) auf eine zulässige (oder korrekte) Menge einschränken (vgl. Abs. 1.3 und Kap. 5).

Konzeptionelles (Datenbank-)Schema

Ein Konzeptionelles Relationales Schema "KS" einer Datenbank "DB" läßt sich schreiben als ein Paar (S, INT) , wobei gilt

(1) S ist eine Schemabeschreibung der Form

$$S = \{ R_i(A_{i1}, \dots, A_{in_i}) \mid i = 1, \dots, m \text{ und } R_i(A_{i1}, \dots, A_{in_i}) \text{ sind Relationenschemata} \};$$

- (2) INT ist eine Menge von Integritätsbedingungen mit $INT = \{int-1, \dots, int-k\}$, die festlegt, daß ein Datenbankzustand $(\underline{R}_1, \dots, \underline{R}_m)$ integer ist, wenn $\bigwedge i (int-i(\underline{R}_1, \dots, \underline{R}_m) = TRUE)$ (verkürzt $INT(db) = TRUE$) gilt.

Ein erster Überblick der Integritätsanforderungen wird in Abs. 1.3. gegeben, ausführlich werden die Möglichkeiten zur Integritätsüberwachung in einem Datenbank-Anwendungssystem in Kapitel 5 behandelt.

Ein Beispiel für ein Konzeptionelles Datenbankschema findet sich im Anhang 1. Es ist das Schema zur Miniwelt eines pharmazeutischen Außendienstes. Dieses Konzeptionelle Schema ist die Grundlage für alle weiteren Beispiele in der Arbeit.

Wie die Konstruktion eines Konzeptionellen Schemas aus der Anwendung heraus erfolgt, das seinen Zweck der Beschreibung der relevanten Sachverhalte einer Anwendung in den zugehörigen Relationen erfüllt, ist noch nicht abgeklärt. Die Diskussion, wie eine solche Konstruktion systematisch erfolgen kann, ist noch nicht abgeschlossen <BZ81>; auch wenn bereits erste Normierungsversuche <ISO82> unternommen werden. Die Untersuchungen über die Art und Weise, in der Sachverhalte einer Anwendung zu erfassen, zu schematisieren sind und letztlich daraus ein Konzeptionelles Schema zu konstruieren ist, wird in der Datenbank-Technologie im Rahmen der "Semantischen Datenmodellierung" (vgl. <VRT82>, <BZ81>) bzw. der "Systematischen Konstruktion von Datenbanken" (vgl. <WO80>, <Wed81>) erörtert.

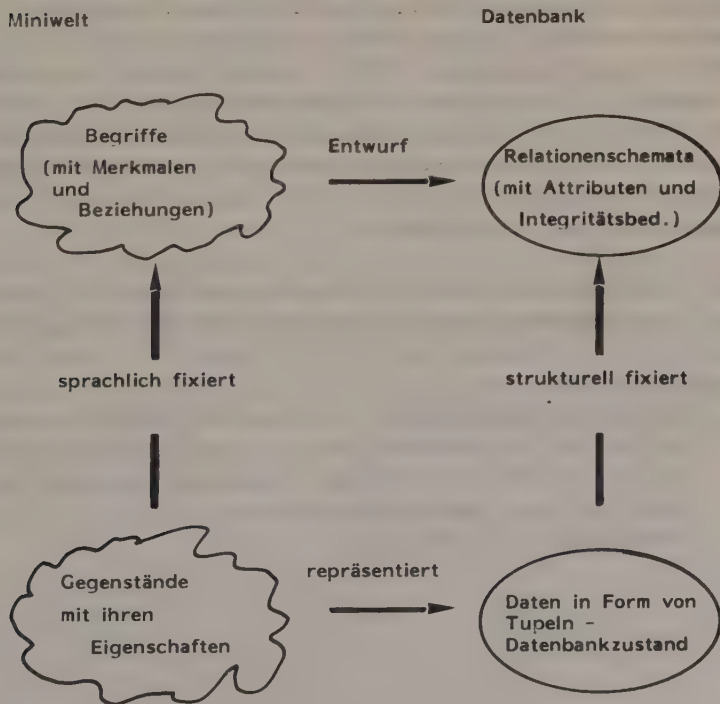
In der "Semantischen Datenmodellierung" wird das Relationale Modell als inadäquat betrachtet, um die Situationen einer Anwendung zu erfassen. Man versucht "höherstehende" Datenmodelle einzuführen, die komplexere Strukturen als

Relationen zulassen. In einem nächsten Schritt wird dann versucht, diese komplexen Strukturen auf das Relationale oder ein anderes Modell abzubilden (<HML78>, S.34). Eine Alternative bietet die Vorgehensweise, die Begriffe direkt in normativer Form auf Relationenschemata zurückzuführen (<Wed81>, S.176ff).

Es bleibt festzuhalten, die anwendungsbezogene Semantik des Konzeptionellen Schemas sind die Begriffe der Miniwelt, ihre Merkmale und Beziehungen.

Abb. 1-1:

(* Beziehung zwischen Miniwelt und Datenbank *)



Einen schematischen Überblick für das Zusammenwirken der Aspekte

- Begriffe der Miniwelt
- Relationenschemata
- Gegenstände der Miniwelt mit ihren Eigenschaften
- Daten in Datenbank

gibt Abb. 1-1.

1.1.2. Sichten der Datenbank

Den Benutzern oder Benutzergruppen der Datenbank sind die Beschreibungen und Darstellungen der Daten zur Verfügung zu stellen, die ihren spezifischen Anwendungssituationen entsprechen (vgl. <TL76>). Daneben ist auch zu verhindern, daß Datenbankbenutzer an Daten aus ihnen fremden Anwendungsteilbereichen gelangen <CGT75>. Beide Aspekte führen zur Forderung der Bildung von (lokalen) Sichten auf die Datenbank, durch die nur ein Ausschnitt der Datenbank einem Benutzer sichtbar wird. (Die Sichten können prinzipiell auch andere Datenstrukturen, als sie im Konzeptionellen Schema beschrieben sind, aufweisen.)

Die Beschreibung und die Ableitung der (lokalen) Sichten einer Datenbank werden in Externen Schemata festgelegt.

Externes Schema

Ein Externes Schema ES ist ein Paar (EX, REP), wobei EX eine Schemabeschreibung (vgl. Konzeptionelles Schema) und REP eine Funktion (Repräsentationsfunktion) ist, die als Definitionsbereich die Datenbank und als Wertebereich die Sicht hat.

Wert der Datenbanksicht

In jedem Datenbankzustand db ist durch die Funktion **REP** aus dem Externen Schema ein Wert "vdb" einer Sicht der Datenbank durch $REP(db)=vdb$ bestimmt.

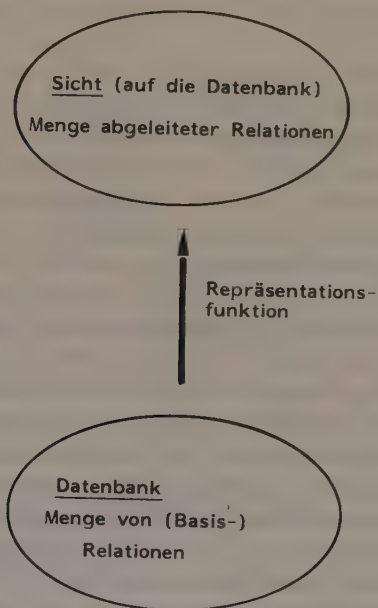
Die Angabe der Repräsentationsfunktion "REP" ist datenmodellspezifisch. Sie kann im Relationalen Modell mit Hilfe der zugehörigen Datenmanipulationssprachen gebildet werden. Eine Sicht auf eine Datenbank besteht aus einer Menge abgeleiteter Relationen aus den (Basis-)Relationen der Datenbank. Im folgenden wird (mit einigen Erweiterungen) QUEL zur Angabe der Ableitungsvorschrift verwendet (vgl. Abs. 3.1).

Über die Bildung von Sichten wurden bereits mehrere Untersuchungen angestellt (vgl. dazu u.a. <Sto75>, <Adi81>, <BS81>, <DB78>, <Lea81>, <Zör82>). Die Problematik liegt dabei in der Umsetzung von Änderungsoperationen auf den Sichten in Operationen auf der Datenbank. Im Abs. 3.1 wird explizit behandelt, wie sich das Sichtenkonzept auf die Strukturierung von Datenbank-Anwendungen auswirkt.

Die Sichten haben eine besondere Bedeutung beim Erstellen von Anwendungssystemen, da sie zur Datenunabhängigkeit vom Konzeptionellen Schema beitragen. Bei Anwendung des Sichtenkonzepts muß sich eine mögliche Veränderung des Konzeptionellen Schemas nicht auf die spezifische Teilanwendung auswirken. Neben der Datenunabhängigkeit ist das Sichtenkonzept die Basis zur Gewährleistung der Datenbank-Sicherheit <Sto75>.

Abb. 1-2 :

(* Zusammenhang Sicht und Datenbank *)



1.2. Ansätze zu einer operationalen Komponente eines Datenbank-Anwendungssystems

Die Miniwelt einer Anwendung ist nicht nur statisch. Nach einem einmaligen Aufnehmen eines Zustands der Anwendung in die Datenbank würde dies bedeuten, daß dieser sich nicht mehr verändern würde. Veränderungen in der Miniwelt der Anwendung sind auf der Datenbank nachzuvollziehen, d.h. es sind entsprechende Operationen vom bzw. auf dem Datenbanksystem bereitzustellen. Von einem Datenbank-Anwendungssystem wird zusätzlich erwartet, daß es auch die Durchführung von Verfahren der Anwendung geeignet unterstützt (<Gra81a>, <Schu81>, <McL79>).

Am Beginn des Einsatzes von Datenbanken stand die Unterstützung von Operationen zur Gewinnung der in der Datenbank abgelegten Information im Vordergrund, d.h. das Ableiten bestimmter Aussagen aus der Datenbank. Mit diesem Ziel wurden mengenorientierte Abfragesprachen entwickelt, darauf aufgebaut wurde dann die mengenorientierte Datenmanipulation (<Cod82>, S.110).

Eine Datenbank stellt mehr als nur eine Beschreibung der Miniwelt der Anwendung dar, da sie als sprachliche Repräsentation von Sachverhalten selbst zum Bestandteil der Miniwelt wird. Dies geschieht genau dann, wenn Vorgänge auf sprachlicher Ebene betrachtet werden und sich somit direkt auf der Datenbank realisieren lassen. Die Operationen auf der Datenbank vollziehen dann nicht nur Geschehnisse der Miniwelt nach, sondern ersetzen diese Vorgänge ganz oder zumindest teilweise. Dies gilt insbesondere für sprachliches Handeln.

Wegen der zunehmenden Bedeutung der Datenbanken für den Aufbau von Anwendungssystemen (vgl. <BZ80>, S.61) entstand die Forderung nach einer Erweiterung der Datenbanksysteme um eine operationale Komponente, in der die Vorgänge einer

Anwendung strukturell erfaßt und in Datenbank-Operationen umgesetzt werden (<Mel81>). Dazu ist es notwendig, entsprechende operationale Strukturen und deren Realisierung auf der Basis elementarer Datenbank-Operationen bereitzustellen (vgl. <Wed79>, <Rol81>, <Bro81>). Die Basis für die Darstellung von Vorgängen aus der Miniwelt ist das Transaktionskonzept <Gra81>. Ergänzend ist ein Kontrollsystem zur Gewährleistung des korrekten Ablaufs der Operationen bzw. Transaktionen und der Integrität der Datenbank aufzubauen (vgl. <FSW81>).

Ziel dieser Arbeit ist es einen Beitrag zur Strukturierung von Vorgängen aus der Miniwelt in Form von Transaktionen und deren Kontrolle zu leisten. Im weiteren dieses einführenden Abschnitts wird kurz auf die verfügbare mengenorientierte Datenbank-Schnittstelle, das Transaktionskonzept und den derzeitigen Stand der Überlegungen für den Aufbau einer operationalen Komponente auf einem Datenbanksystem eingegangen.

1.2.1. Mengenorientierte Datenbank-Änderungsoperationen

Zur Realisierung von Vorgängen aus der Miniwelt auf der Datenbank stehen in den Datenmodellen verschiedene elementare Datenbank-Operationen zur Verfügung. Es wird dabei prinzipiell zwischen informationsgewinnenden Operationen (Leseoperationen), die den Zustand der Datenbank nicht verändern, und Änderungsoperationen (Schreiboperationen) unterschieden. Für die Realisierung von Leseoperationen steht im Relationalen Modell die Relationale Algebra <Pir82> bzw. gleichmächtige Sprachen wie SQL <Cha76> oder QUEL <Sto75> zur Verfügung. Im folgenden der Arbeit wird die Notation der elementaren Leseoperationen entsprechend der Konventionen von QUEL vorgenommen (vgl. <RT82>).

Die Änderungsoperationen für das Einfügen, Ändern und Löschen von Tupelmengen aus Relationen der Datenbank bzw. einer Sicht der Datenbank können ebenfalls als mengenorientierte Grundoperationen angegeben werden. Die Syntax der Änderungsoperationen wurde zu QUEL geringfügig im Vorgriff auf Kapitel 2.2 verändert. Die Notation der einzelnen Operationen ist dennoch fast selbst erklärend.

Für die Einfügeoperation wird vereinbart:

APPEND TO <relation> (<target-list>)

WHERE <qualification>;

Durch diese Änderungsoperation lassen sich in eine Relation mit dem Namen "relation" bezüglich den angegebenen Attributen "target-list" eine Menge von Tupeln, die sich entweder aus einer anderen Relation qualifiziert haben oder explizit angegeben wurden, eintragen.

Beispiel:

(1) Tupelmenge aus einer anderen Relation.

RANGE OF a IS ARZT;

APPEND TO EINLADUNG

(Anrede=a.Anrede,Titel=a.Titel,
Vorname=a.Vorname,Name=a.Name,
Adresse=a.Adresse)

WHERE a.RPM-KreisNr=75;

Die Abweichung zu QUEL besteht darin, daß explizit eine Zuordnung zu der einzelnen Attributen erfolgt; in QUEL dagegen die Reihenfolge aus der "target-list" als Zuordnung angenommen wird.

(2) Einfügen einzelner Tupeln

APPEND TO EINLADUNG

(Anrede="Herr",Titel="Dr. med",
Vorname="Axel",Name="Krause",
Adresse="Schillerstr 13 8800 Ansbach")

Für das Löschen von Tupelmengen aus Relationen der Datenbank steht folgendes Kommando zur Verfügung

DELETE <range-var> WHERE <qualification>;

Beispiel:

RANGE OF a IS ARZT;

DELETE a WHERE a.Klasse="C" AND Geb-Datum<"1.1.1920";

Die Anweisung zum Ändern von Relationen ergibt sich entsprechend der Anweisung zum Einfügen.

REPLACE <range-var> (<target-list>)

WHERE <qualification>;

Beispiel:

RANGE OF a IS ARZT;

REPLACE a (a.Klasse="B".)

WHERE a.Klasse="N" AND Fach="Internist";

Bemerkung: Der Bindestrich in den Attributnamen ist in QUEL nicht zulässig. Aus drucktechnischen Gründen steht er hier für den erlaubten Unterstrich.

Beim Ausführen einer dieser elementaren (wenn auch mengenorientierten) Änderungsoperationen ist die Einhaltung der Datenbankintegrität zu beachten (vgl. dazu auch Abs. 1.3). Eine Prüfung, ob letztlich die Integrität der Datenbank verletzt ist, kann aber erst am Ende einer Folge solcher elementarer Datenbank-Operationen entschieden werden. Dies ist dadurch bedingt, daß diese elementaren Datenbank-Operationen nicht ausreichen, um komplexe Vorgänge der Miniwelt nachzubilden. Sie verletzen sowohl die semantische Integrität als auch strukturelle Eigenschaften des Datenmodells, wie z.B. referentielle Beziehungen die zwischen Attributen mehrerer Relationen (vgl. dazu auch Abs. 1.3) bestehen können. In der Regel ist eine Folge von Datenbank-Operationen zur Realisierung eines Vorgangs aus der Miniwelt der Anwendung notwendig. Innerhalb einer solchen Folge befindet sich folglich die Datenbank in einem nicht integeren bzw. inkonsistenten Zustand.

Bezüglich Datenbank-Änderungsoperationen ist somit eine höhere Schnittstelle zum Anwender hin erforderlich. In der Regel wird diese durch das Bereitstellen spezifischer Anwendungsprogramme realisiert (vgl. <Gra77>, <Cha80>). Als erstes ist zu fordern, daß die Operationen, die durch solche Programme realisiert werden, dem Anwender als ein einheitliches System zur Verfügung gestellt werden. Wie sich aus der Arbeit ergeben wird, ist es aber nur bedingt möglich, alle potentiell möglichen Vorgänge einer Anwendung in der Form vorgegebener Programme zu realisieren. Es führt zu einem erheblichen Aufwand und würde ein kreatives Handeln verhindern (vgl. Kap. 4).

Für den Aufbau einer operationalen Komponente eines Datenbanksystems gilt es, eine Systemstruktur für die Einbettung von Anwendungsoperationen zu entwickeln, die die Formulierung komplexer Anwendungsoperationen als atomare Einheiten zuläßt, unterstützt und verwaltet. Dem Anwender ist dabei für seine Operationen eine adäquate Schnittstelle

zur Verfügung zu stellen.

1.2.2. Das Konzept der Transaktion

Das Konzept der Datenbank-Transaktion <EGLT76> (im weiteren kurz als Transaktion bezeichnet) wurde im Rahmen der Entwicklung des System R (<Ast76>, <Bla79>, <Ast80>) in seiner paradigmatischen Form eingeführt. Es ist als das Grundkonzept für alle Datenbank-Bearbeitungen anzusehen. Eine Transaktion besteht aus der Ausführung einer Folge elementarer Datenbank-Operationen, die zu einer Einheit zusammengefaßt werden.

Der Grundgedanke des Transaktionskonzepts drückt sich nach Gray <Gra81> in folgenden Kriterien aus. Eine Transaktion ist

- konsistent (consistent),
- atomar (atomic),
- dauerhaft (durable).

Konsistenz

Die Forderung nach Konsistenz zerfällt in drei Teilaspekte. Der erste bezieht sich auf die Aufrechterhaltung der Datenkonsistenz oder -integrität. Eine Transaktion überführt die Datenbank aus einem korrekten (widerspruchsfreien) in einen wiederum korrekten Zustand. Der Konsistenzbegriff umfaßt dabei erstens die Gültigkeit der Integritätsbedingungen auf der Datenbank, wie sie im Abs. 1.3 und Kap. 5 noch erläutert werden. Die Integritätsbedingungen müssen vor und nach Ausführung einer Transaktion erfüllt sein.

Der zweite wesentliche Aspekt ist, falsche Einträge und Auswertungen zu verhindern, die durch Effekte der parallelen Ausführung von Transaktionen auftreten können (<EGLT76>, <Gra77>, S.432). Für den Ablauf einer Transaktion ist zu gewährleisten, daß er sich logisch völlig isoliert betrachten läßt und keine Inferenzen mit anderen Transaktionen aufweist. Die Transaktionen müssen serialisierbar sein. Die Vermeidung der gegenseitigen ungewollten Beeinflussung wird auch als Aufrechterhaltung der Ablaufintegrität bezeichnet. Von Härder/Reuter (<HR83>, S.44) wird die Isolation als eigenständiges Transaktionskriterium eingeführt.

Aus dem Kriterium der Isolation ergibt sich das Problemfeld der Synchronisation von Transaktionen. Dieser Aspekt wird im weiteren der Arbeit ausgeklammert, da hierfür hinreichende Mechanismen auf der Ebene der Datenbank-einträge vorhanden sind (einen Überblick gibt <BHR80>, <GM82>). Die Problematik einer "logischen" Synchronisation von Transaktionen, das heißt das apriori Festlegen von Reihenfolgen von Transaktionen, erscheint im Bereich der klassischen Datenbank-Anwendungsgebiete nicht relevant zu sein. Eine aus der Anwendung erforderliche Reihenfolge von Transaktionen kann, falls erforderlich, über den Datenbankzustand (Zustandsvariable in der Datenbank) festgelegt werden. Zu erwähnen ist, wird der Einsatz von Datenbanken für Prozeßsteuerungen betrachtet, ist die Notwendigkeit, die Reihenfolge von Transaktionen festzulegen offensichtlich. Die Synchronisation zwischen Transaktion sind dann u.U. die Mechanismen auf der Ebene der Datenbank nicht mehr hinreichend.

Der dritte Aspekt der Konsistenz ist, daß die Ausführbarkeit der Transaktion selbst gewissen Beschränkungen unterliegt, d.h., daß eine Transaktion nur in bestimmten Zuständen der Datenbank ausführbar ist. Die Einhaltung dieser Bedingungen wird auch als Wahrung der Übergangs-

oder Transaktionsintegrität bezeichnet (vgl. <EC75> und <Web81>, S.7).

Die Aspekte der Datenbank- und der Transaktionsintegrität werden in Kapitel 5 der Arbeit weiter vertieft.

Atomarität

Die Forderung nach Atomarität erzwingt die vollständige Ausführung einer Transaktion oder das Rücksetzen aller ihrer Effekte auf der Datenbank. Die Transaktion wird für den Benutzer zur unteilbaren Einheit, die nach dem "Alles-oder-Nichts"-Prinzip vom Datenbanksystem zu verwalten ist. Diese auf der Ebene des Benutzers atomare Einheit besteht aus einer Folge elementarer Datenbank-Operationen. Das Datenbanksystem hat dafür zu sorgen, daß bei Abbruch einer Transaktion der Status quo ante der Datenbank wiederhergestellt wird. (In vielen realen Systemen fehlt noch ein diesbezüglicher Mechanismus, vgl. <SB83>).

Zur Gewährleistung der Atomarität von Transaktionen wurden eine ganze Reihe von Verfahren entwickelt, die bei gewünschten, wie auch unerwünschten Abbrüchen von Transaktionen den "alten" Zustand der Datenbank wiederherstellen (vgl. <Reu81>).

Das Kriterium der Atomarität ist als die zentrale strukturelle Eigenschaft im Hinblick auf den Entwurf von Transaktionen anzusehen, wie in den folgenden Abschnitten noch deutlich wird.

Dauerhaftigkeit

Unter der Dauerhaftigkeit einer Transaktion ist zu verstehen, daß die Wirkung einer ordnungsgemäß abgeschlossenen Transaktion beständig ist. Sie kann in ihrer Wirkung durch keinerlei Systemeinfluß (z.B. Zusammenbruch)

verloren gehen bzw. ungeschehen gemacht werden. Eine Annullierung kann nur mittels Kompensation, d.h. durch das Ausführen einer anderen Transaktion erreicht werden, die den ursprünglichen Datenbankzustand wieder herstellt. Es ist evident, daß nicht alle Transaktionen auf diese Weise kompensierbar sind, wenn die Integrität der Datenbank gewahrt bleiben soll. Andere Transaktionen können bereits den neuen Zustand referenziert haben. Die Dauerhaftigkeit umfaßt somit, daß beim ordnungsgemäßen Abschluß einer Transaktion ihre Ergebnisse für alle anderen Transaktionen als gültig erkannt werden. Das Erklären der Gültigkeit erfolgt durch das sog. Transaktions-Commitment (<Gra80>, S.285). Der durch eine Transaktion erzeugte Zustand ist korrekt und darauf ist zum Zeitpunkt des Abschlusses einer Transaktion Verlaß.

Eine Transaktion T kann somit als eine Abbildung eines integeren in einen wiederum integeren Datenbankzustand notiert werden,

```
T : DATABASE -----> DATABASE
und
INT(db-vorher) < T > INT(db-nachher)
```

Die Ausführung einer Transaktion T kann auch als eine Folge von Ausführungen elementarer Datenbank-Operationen aufgefaßt werden.

Bemerkung: (Die Gray'sche Aufzählung der wichtigsten Eigenschaften von Transaktionen könnte man um die Forderung ergänzen, daß sie auch den Schutz der Datenbank vor mißbräuchlicher Verwendung unterstützen (vgl. dazu <Min76>). Nur berechtigten Benutzer sollte es möglich sein, bestimmte Transaktionen zu aktivieren bzw. auszuführen (vgl. Kap. 6). Der Schutz der Datenbank auf der Basis von Transaktionen wurde ebenfalls im Rahmen des System R untersucht <Cha78>.)

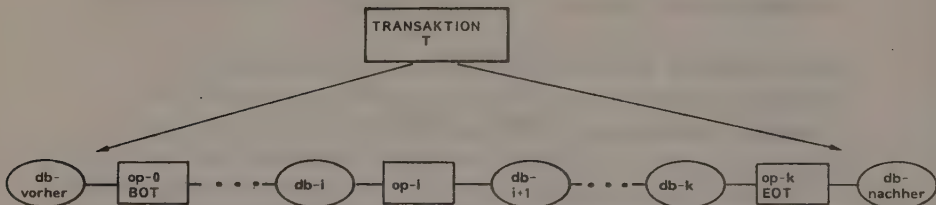
Analog zu Gray <Gra80> kann man dann schreiben:

$$T = \langle t, \text{op-}i, \text{db-}i \mid i = 0, \dots, k \rangle$$

wobei t eine Transaktionskennung und $\text{op-}i$ eine mögliche Operation im Datenbankzustand $\text{db-}i$ wiedergibt. Die Operation $\text{op-}0$ ist als Startoperation zur Kennzeichnung des Beginns der Transaktion (BOT) und $\text{op-}k$ als Commit für das Ende der Transaktion (EOT) zu interpretieren. Diese beiden Datenbank-Operationen sind als Klammern der Transaktion anzusehen, um sie als atomare Einheit zu kennzeichnen.

Abb. 1-3:

(* Transaktion aus elementaren DB-Operationen *)



Zusammenfassend besteht die operationale Komponente eines Datenbank-Anwendungssystems aus einer Menge von Transaktionen, die im folgenden auch als Transaktionssystem bezeichnet werden. Es soll nicht verschwiegen werden, daß es bei der Anwendung des Transaktionskonzepts einige Schwierigkeiten, z.B. bei langlebigen Transaktionen, auftreten können. Weitere Hinweise hierzu finden sich in <Gra81>.

1.2.3. Ansätze zum Transaktionsentwurf und zum Aufbau eines Transaktionssystems

Im Rahmen der Datenbank-Technologie kommt dem Transaktionsentwurf und der -verwaltung analog zum Entwurf des Konzeptionellen Schemas der Datenbank (vgl. dazu <Wed79a>) eine verstärkte Aufmerksamkeit zu. Beides, Entwurf und Verwaltung, sind eng mit einander verknüpft. Nur in Kenntnis der Grundstrukturen von Transaktionen ist es möglich, ein geeignetes Transaktionssystem aufzubauen, das den Bedürfnissen einer Anwendung entspricht.

Der Gedanke des Aufbaus eines Transaktionssystems findet sich bereits im hierarchischen Datenbanksystem IMS der IBM (vgl. <ALM82>). Natürlich fehlen dort all jene Transaktionsanforderungen, wie sie im vorherigen Abschnitt skizziert wurden bzw. wurden sie dem Anwendungsprogrammierer in seine eigene Verantwortung übertragen. Dieses Vorgehen ist letztlich unbefriedigend. Auch im Rahmen der Relationalen Datenbanksysteme wurden Versuche unternommen, eine Transaktionsverwaltung in das Datenbanksystem zu integrieren oder auf dem Datenbanksystem aufzusetzen. Ansätze finden sich z.B. im System R (<Esw76>, <Gra77>, <Cha80>), ENCOMPASS <Schu81>, IDM (<GEI82>), PASCAL/R (<RS81>) oder in LIDAS (<RRUZ82>). Die bereitgestellten Strukturen entsprechen aber noch nicht den Anforderungen, die an ein transaktionsorientiertes Anwendungssystem zu stellen sind.

Das Problem wurde konzeptionell aufgegriffen durch die Notwendigkeit zur Bereitstellung höherer Operationen für den Benutzer eines Datenbanksystems, wobei man sich nicht auf relationale Datenbanksprachen beschränken wollte (<Cha78>, <Wed79>, <Mel81>, <RZ82>). Dies betrifft insbesondere Datenbank-Anwendungen mit einem hohen Anteil standardisierter, immer wiederkehrender Änderungsoperationen auf der Datenbank. Zusätzlich entstand auch das Bedürfnis einer Erhöhung des Schutzes der in der Datenbank gespeicherten Information

sowie die Notwendigkeit, diesen auf operationaler Basis zu gewährleisten (<Min76>, <Cha78>, <Sch182>). Beide Aspekte führten zu einer Reihe von Überlegungen aus denen sich die Grundstrukturen für transaktionsorientierte Datenbank-Operationen entwickeln lassen.

Bevor die "Modellierung" von Transaktionen untersucht werden kann (wie dies in <FR78>, <Bro81> geschieht), sind zuerst die Strukturen zur Darstellung von Transaktionen zu schaffen. Die Wechselwirkung zwischen Datenbank- und Transaktionssystem zum Aufbau eines Anwendungssystems skizzieren Abb.1-4 und 1-5. Die Transaktion zeichnet sich dabei als Realisierung einer höheren Datenbankschnittstelle aus.

Abb. 1-4:

- (* Transaktionssystem als weitere Schicht im Datenbank-Anwendungssystem *)
(in Erweiterung der Datenbanksystemarchitektur nach <Reu81>, S.33)

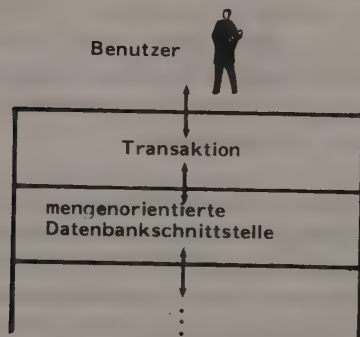
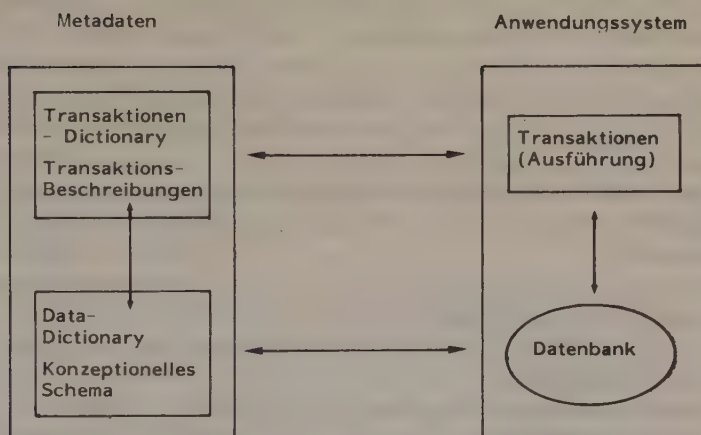


Abb. 1-5: (* Trennung zwischen Meta- und Nutzdaten *)



1.3. Integritätskontrolle - Wechselwirkungen zwischen Transaktionen und Datenbank

Die Integritätsanforderungen innerhalb eines Datenbank-Anwendungssystems sind vielfältig. Sie umfassen sowohl die Ausführung von Transaktionen, die Beschreibung und Abspeicherung der Daten, als auch die Beziehung der Datenbank zur Miniwelt. Es gibt deshalb eine ganze Reihe von Klassifikationen der Integrität und den zugehörigen Bedingungen (vgl. z.B. <EC75>, <Thu80>, <Web81>, <Wed83a>). Prinzipiell können folgende Klassifikationsmöglichkeiten unterschieden werden; die je nach Zwecksetzung sinnvoll erscheinen,

- (1) nach der (Schema-)Ebene, auf die sich die Integritätsanforderungen beziehen,
(Man unterscheidet physische, strukturelle und semantische Integrität.)
- (2) nach den Kriterien operational oder objektbezogen,
(Es ergibt sich eine operationale und statische Integrität, aus der der Zuordnung der Bedingungen zu den (Trans-)Aktionen oder zu den Daten zur Kontrolle der Korrektheit des Übergangs oder des Zustands.)
- (3) nach der Relevanz für die Rekonstruktion der Begriffe aus der Anwendung,
(Beim Entwurf sind konstitutive und regulative Bedingungen zu unterscheiden, ob sie für die Anwendung konstitutiv, damit invariant, oder regulativ, damit modifizierbar, sind.)
- (4) nach dem Umfang der betroffenen Objekte der Datenbank,
(Die Unterteilung z.B. interrelational, intrarelational, etc. ist bei der Bereitstellung von Kontrollmechanismen und Untersuchungen zu deren Effizienz relevant.)

Diese Einteilung ist nicht disjunkt, sondern beschreibt verschiedene Gesichtspunkte der Integrität in einer Datenbank-Anwendung.

So ist z.B. die Bedingung: "Ein Arztbesuch darf nur dann eingetragen werden, wenn der Arzt im Bezirk des Außendienstmitarbeiters wohnt." (vgl. dazu auch das Beispiel im Anhang.)

- semantisch,
(da die Bedingung anwendungsbezogen auf die Miniwelt ist)
- operational,
(da die Bedingung die Ausführbarkeit der "Operation" Arztbesuchseintrag festlegt)
- regulativ,
(da die Bedingung nicht konstitutiv für den Arztbesuchseintrag ist, sondern ohne Schaden der Begriffsstrukturen geändert werden kann)
- interrelational
(da mehrere Relationen - ARZT, MITARBEITER, ARZTBESUCH - von dieser Bedingung betroffen sind).

Für die weiteren Betrachtungen sind die Integritätsbedingungen nach Punkt (1) und (2) von besonderem Interesse. Deshalb zu diesen Punkten noch einige erläuternde Bemerkungen.

(1) statische oder objektbezogene Integrität
(auch Datenintegrität)

Die statische Integrität gewährleistet einen korrekten Zustand der Datenbank. Die statische Integrität ist somit invariant gegenüber Veränderungen und ist vor und nach jeder Transaktion erfüllt. Die statische Integrität erstreckt sich sowohl auf die Speicherung als auch auf die Darstellung der Daten, sowie auf deren Bedeutung, d.h. auf die zugeordneten Objekte und Begriffe der Miniwelt.

(2) operationale oder aktionsbezogene Integrität

Die operationale Integrität gewährleistet den korrekten **Übergang**. Sie bezieht sich dabei auf den Übergang selbst. Sie gibt an, wann ein Übergang im Sinne der Datenbank-Anwendung korrekt ist. Sie ist damit transaktionsorientiert und umfaßt die Überwachung der Übergänge.

Arten von Integrität

Für das weitere Arbeiten sind diese beiden Aspekte der Integrität um die Differenzierung nach der Ebene, in der sie anzusiedeln sind, zu ergänzen..

(1) physische Integrität

- in der objektbezogenen Betrachtung

Vom Datenbanksystem ist die korrekte Umsetzung der logischen Datenstrukturen in die physischen Datenstrukturen aufrecht zu erhalten. Die Abbildung der Daten, wie sie der Benutzer kennt, auf die maschineninterne Darstellung ist zu jeder Zeit konsistent bzw. im Fehlerfalle rekonstruierbar. Für diese Art von Integrität existieren in den Datenbanksystemen entsprechende Mechanismen (vgl. dazu <Reu81>). Es ist die Korrektheit der Datenrepräsentation über die Ebenen eines Datenverwaltungssystems zu garantieren. (Im folgenden auch als physische Datenintegrität bezeichnet).

- in der aktionsbezogenen Betrachtung

Analog zur Datenrepräsentation ist die Korrektheit der Implementation der Datenmanipulationsoperationen zu gewährleisten. Diese Art der Integrität gilt als gesichert, wenn man das Datenverwaltungssystem und somit auch die zur Verfügung gestellten Operationen als korrekt annimmt.

Als Möglichkeiten zur Sicherung der physischen Integrität einer Datenbank ist das physische Recovery zu nennen, indem physische Sicherungskopien von der Datenbank genommen

werden (<Reu81>, S.154).

(2) strukturelle Integrität

- in der objektbezogenen Betrachtung

(Auch als strukturelle Datenintegrität bezeichnet)

Jedem Datenmodell liegt eine gewisse Struktur zugrunde, in der die Begriffe der Miniwelt erfaßt werden. Das Datenmodell stellt Restriktionen an die Darstellung der Daten, die einzuhalten sind. Das Relationale Datenmodell kennt z.B. zur Darstellung von Begriffsbeziehungen die referentielle Integrität <Dat81a> oder Fremdschlüsselbeziehung. Solche Beziehungen müssen in einem korrekten Zustand der Datenbank gelten.

Die strukturellen Restriktionen des Relationalen Datenmodells werden in Abs. 5.3.1 kritisch reflektiert.

- in der operationalen Betrachtung

Die strukturelle Integrität bezogen auf die Operationen auf einer Datenbank leiten sich aus den Transaktionskriterien ab (vgl. Abs. 1.2.2). Die Gray'schen Kriterien bilden die entscheidenden Strukturierungsprinzipien. Die strukturelle, operationale Integrität wird im folgenden auch als Ablaufintegrität bezeichnet. (Bei Härder <Här78> wird die Bezeichnung "Ablaufintegrität" für die Sicherung der Isolation verwendet. Im weiteren soll die Wahrung der Atomarität und der Dauerhaftigkeit mit eingeschlossen werden.)

Es ist auszuschließen, daß sich Transaktionen wechselseitig beeinflussen. Zum anderen ist durch die Forderung nach Atomarität der Transaktion eine Folge von Datenbankoperationen geklammert, die als eine Einheit für die Datenbank-Anwendung zu sehen sind. Die operationale strukturelle Integrität hat für die Sicherung der Gray'schen Transaktionskriterien Sorge zu tragen.

(3) semantische Integrität

- in der objektbezogenen Betrachtung

Unter dem Begriff der semantischen objektbezogene Integrität werden eine ganze Reihe von Integritätsanforderungen subsumiert (z.B. auch strukturelle Eigenschaften <Dat81a>). Die semantische objektbezogene Integrität (kurz semantische Datenintegrität) umfaßt in ihrem ursprünglichen Verständnis <EC75> die Beschränkung der in einem Konzeptionellen Schema fixierten möglichen Datenbankzustände auf eine "sinnvolle" Teilmenge. Mit Hilfe von semantischen Integritätsbedingungen versucht man die Datenbankzustände auf jene einzuengen, deren dargestellte Sachverhalte in der Miniwelt auch wirklich auftreten können. Die Angabe von Integritätsbedingungen zur Aufrechterhaltung der semantischen Datenintegrität kann damit nur eine Teilmenge der unzulässigen Zustände ausschließen. Die Frage, ob ein dargestellter Sachverhalt fingiert, d.h. die Aussagen über ihn falsch sind, oder ob der Sachverhalt wirklich ist, ist durch die semantische Integrität nicht entscheidbar.

- in der operationalen Betrachtung

Die semantische operationale Integrität kann man als Übergangsintegrität auffassen. Sie gewährleistet die Korrektheit des Übergangs, realisiert durch eine Transaktion. Eine Transaktion ist nur unter bestimmten, transaktionsspezifischen Vorbedingungen ausführbar. Ebenso ist zu prüfen, ob von einem gegebenen Zustand der Datenbank der erreichte Endzustand bezüglich der Spezifikation der ausgeführten Operation korrekt ist. Die Übergangsintegrität entspricht den in der Programmierung üblichen Vor- und Nachbedingungen.

Diese erste grobe Klassifikation der Integrität läßt erkennen, daß für den Aufbau eines Anwendungssystems auf Transaktionsbasis die physische Integrität (sowohl statisch wie operational) zurückgestellt werden kann, da sie in un-

teren Schichten eines Datenbanksystem gewährleistet wird. Für ein transaktionsorientiertes Anwendungssystem steht die semantische und strukturelle Integrität im Vordergrund.

Die aufgezählten Integritätsarten sind um eine deontische Integrität zu ergänzen (vgl. Kap. 5), die es gestattet den "Gebrauch der Datenbank" zu kontrollieren. Sie hat einen vorliegenden Zustand der Datenbank bzw. Miniwelt im Hinblick auf einen intendierten Zustand abzugleichen und zu beurteilen. Operational gesehen sind die Transaktionen hinsichtlich der Kriterien erlaubt, verboten und geboten zu beurteilen und zu überwachen.

Integritätskontrollsphäre

Die Kontrollmechanismen zur Realisierung der integritätssichernden Maßnahmen erfordern, daß zu unterscheiden ist, ob die Daten nur gelesen oder auch verändert werden. Der lesende Zugriff mehrerer Transaktionen auf ein Datenobjekt kann im sog. Teilhabe-Modus (shareable mode) erfolgen, während der Schreibzugriff unter gegenseitigen Ausschluß (exclusive mode) zu geschehen hat (vgl. <Gra77>). Die Daten, auf die eine Transaktion in der Datenbank zugreift, spannen eine sog. Integritätskontrollsphäre auf.

Die Idee der Bildung von Kontrollsphären (sphere of control) geht auf Davies (in <Dav72>, <Dav78>) zurück. Im Vordergrund stand dabei die Behandlung der Recovery von Transaktionen, was auf der logischen Ebene der Wahrung der Atomarität entspricht. Die Denkweise in Kontrollsphären eignet sich aber auch zur Darstellung der Zusammenhänge in einem Datenbank-Anwendungssystem. Durch dieses Konzept lassen sich die Wechselwirkungen zwischen Datenbank, Transaktion und Benutzer erfassen.

Die Integritätskontrollsphäre einer Transaktion umfaßt alle Daten der Datenbank, auf die sie direkt (durch die Ausführung elementarer Datenbank-Operationen) oder indirekt (durch Prüf- oder Folgeoperationen, die zur Überwachung der Transaktion notwendig sind,) zugegreift. Die Kontrollsphären sind die (logischen) Grenzen einer Transaktion.

Man kann die Integritätskontrollsphären einer Transaktion oder Operation einteilen in:

- direkte Schreibkontrollsphäre
- direkte Lesekontrollsphäre
- indirekte Schreibkontrollsphäre
- indirekte Lesekontrollsphäre.

Eine formale Festlegung des Begriffs der Kontrollsphäre findet sich in Abs. 5.2 als Menge aller "betroffenen Objekte" einer Operation, die zu überwachen ist.

Die Integritätsbetrachtungen bezogen sich auf die Datenbank und die Transaktionen. Es wird dabei der Grundsatz verfolgt, dem System soviel wie möglich an Kontrolle zu übertragen und möglichst wenig in die Verantwortung der Benutzer der Datenbank zu stellen. Man könnte den Begriff der Integrität noch weiter fassen, indem man die Miniwelt mit einbezieht. Es ist offensichtlich, daß sich auch "Integritätsverletzungen" in der Miniwelt der Anwendung ergeben können. Diese entziehen sich jedoch bis auf die Kontrolle der Eingabe- und Ausgabeparameter der Überwachung durch das Datenbank- bzw. Transaktionssystem.

Die Bestimmung der Integritätskontrollsphären ist im Gegensatz zur Recovery und zur Isolation von Transaktionen (siehe auch <Reu81>) noch nicht erschöpfend behandelt. Die Überwachung der semantischen und der strukturellen Integrität wirkt unmittelbar in das Anwendungssystem hinein, da z.B. die Festlegung der semantischen Integritätsbedingungen anwendungsspezifisch ist.

Die Problematik der Integritätskontrolle wird in Kap. 5 auf der Basis der in Abs. 2.2 und Kap. 4 präzisierten Transaktionsstrukturen und -systems im Hinblick auf eine umfassende Gestaltung eines Datenbank-Anwendungssystems behandelt.

Die ebenfalls wichtigen Kontrollsphären der Synchronisation und Recovery werden in dieser Arbeit nicht weiter ausgeführt, da sie primär das Datenverwaltungssystem selbst betreffen. Eine Untersuchung der logischen Abhängigkeiten in der Reihenfolge und der Möglichkeit der Kompensation von Transaktionen bedarf allerdings noch ausführlicher Untersuchungen (vgl. <Gra81>).

Betrachtet man das Datenbank-Anwendungssystem als Ganzes, so ist die Aussage eine Datenbank befinde sich in einem korrekten oder integeren Zustand, wenn alle ihre Integritätsbedingungen erfüllt sind, zwar formal elegant faßbar. Das Problem der Korrektheit wird jedoch auf die Konstruktion von Integritätsbedingungen verlagert, anstelle korrekte Transaktionen zu konstruieren.

2. Rekonstruktion und Struktur von Transaktionen

Wie bereits dargestellt wurde, bilden die Transaktionen die Einheiten für den Aufbau einer operationalen Komponente eines Datenbank-Anwendungssystems. Die Struktur der Transaktionen resultiert aus ihrem semantischen und pragmatischen Bezug zu den Geschehnissen bzw. Verfahren der Miniwelt. Letztlich leitet sich aus dieser Beziehung die Gesamtstruktur des Anwendungssystems ab.

Das Aufweisen von konkreten Geschehnissen einer Miniwelt und deren Übertragung in die Form von Transaktionen ist nur möglich, wenn entsprechende Strukturen für die Gestaltung von Transaktionen vorhanden sind. Deren Bereitstellung ist das Anliegen dieses Kapitels.

Aufzuzeigen ist, wie sich aus der Miniwelt die Struktur von Transaktionen rekonstruieren läßt (Abs. 2.1) und welche Strukturen in einem Datenbank-Anwendungssystem für ihre adäquate Repräsentation als Transaktionen notwendig werden (Abs. 2.2).

2.1. Rekonstruktion der Transaktionsstrukturen aus den Handlungsstrukturen

Es wird dargestellt, wie Geschehnisse bezüglich einer Datenbank-Anwendung umzustrukturieren sind und aus welchen automatisierbaren Grundkomponenten und -strukturen sie aufgebaut sind. Als Konsequenz wird u.a. dabei deutlich, daß das Transaktionskonzept zu erweitern und die Transaktionsumgebung mit in die Überlegungen einzubeziehen ist.

Die Betrachtungen bezüglich der Geschehnisse in der Miniwelt können dabei eingeengt werden auf Handlungen, wie sie in der Wissenschaftstheorie definiert werden (z.B. <LS75>, S.152, <Kam80>, S.97). Das Ausklammern von Naturprozessen und Verhalten ist darin begründet, daß letztlich nur Handlungen im engeren Sinne automatisierbar, d.h. in diesem Fall dem Datenbank-Anwendungssystem übertragbar sind.

(Um Mißverständnissen vorzubeugen, die Überwachung technischer, biologischer, physikalischer Prozesse und deren Steuerung sind Handlungen. Die Beschränkung grenzt jedoch die Simulation von Prozessen und Verhalten aus.)

Um die Strukturen zur Darstellung von Handlungen und ihrer Umgebung konstruieren zu können, wird zuerst der Bezug zum Handlungsbegriff, wie er in der allgemeinen Handlungstheorie (vgl. z.B. <Res67>, <Wri79>, <Len80>, <Dav80>) verwendet wird, hergestellt. Dies eröffnet die Möglichkeit die Struktur von Handlungen und ihrer Umgebung zu analysieren.

Die folgenden Abschnitte befassen sich somit zum einen mit dem Aufweisen der Grundstrukturen von Handlungen und zum anderen mit der Reduktion dieser Strukturen auf die wesentlichen Komponenten hinsichtlich ihrer Übertragbarkeit auf ein Datenbank-Anwendungssystem.

Ausgehend von der Miniwelt der Anwendung sind die charakteristischen strukturellen Komponenten von Handlungen herauszuarbeiten, um Handlungen in der Form von Transaktionen darstellbar zu machen. Es geht dabei nicht darum, die volle Schönheit der Handlungstheorie kulturwissenschaftlicher Provenienz <Schw76> nachzubilden. Es sind nur jene Handlungsspezifika zu explizieren, die im Rahmen einer Datenbank-Anwendung automatisierbar bzw. unterstützbar sind.

2.1.1. Erste Bemerkungen zum Begriff "Handlung"

Folgt man Kambartel (<Kam80>, S.97), so sind Handlungen nicht einfach Geschehnisse, die dem Handelnden einfach widerfahren, sondern durch ihn zustande gebracht werden. Dies ist der Standpunkt der "Intentionalisten", der davon ausgeht, daß Handlungen stets mit einer Intention verbunden sind. Es besteht ein logischer Zusammenhang zwischen der konkreten Handlung und der Absicht des Handelnden. Ohne auf weitere Argumente für diese Position einzugehen und andere philosophische Meinungen gegenüberstellen zu wollen (vgl. dazu <Len78>), wird im Folgenden diese These geteilt. Als Rechtfertigung im Hinblick auf ein Datenbank-Anwendungssystem mag genügen, daß nur Handlungen für die Automatisierung von Interesse sind, welche eine Intention verfolgen, die an den Zwecksetzungen der Anwendung ausgerichtet ist. Nicht intentional bestimmte Handlungen, falls solche existieren sollten, sind damit für den betrachteten Bereich der Datenbank-Anwendungssysteme nicht relevant.

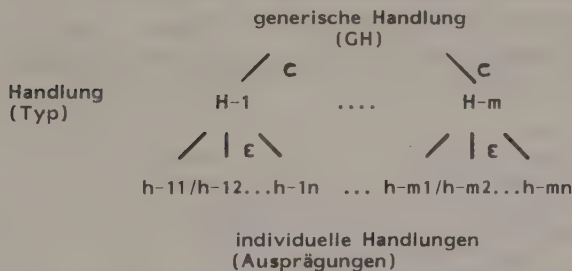
Zur Begriffsklärung ist zuerst die zwischen einer individuellen (oder konkreten) Handlung (<Wri77> S.2) und dem Handlungstyp zu unterscheiden. Man sagt (völlig analog zu den Objekttypen <Wed81>), eine individuelle Handlung h fällt unter einen Handlungstyp $h \in H$, wenn sie die den Typ charakterisierenden Merkmale erfüllt.

Durch einen Handlungstyp wird extensional eine Menge individueller Handlungen bestimmt. Die Beziehung zwischen Handlungstyp und konkreter Handlung ist eine Subsumtion. Dagegen abzugrenzen ist die Beziehung zwischen Handlungstypen, die eine Subordination darstellt. Ein Handlungstyp ist in einer anderen (generischen) Handlung enthalten, wenn alle konkreten Handlungen des einen Tys auch unter den generischen Typ fallen.

Zur Vereinfachung der Sprechweise wird, der Literatur <Wri77> folgend, mit Handlung in der Regel der Handlungstyp und die konkrete Handlung auch als Handlungsausprägung oder -aktualisierung bezeichnet.

Abb. 2-1.:

(* Zusammenhang zwischen Handlungstypen und -ausprägungen *)



Einer konkreten Handlung spricht man gewisse Eigenschaften zu. Sie entscheiden über die Einordnung in einen Handlungstyp, indem sie dessen Merkmale erfüllen. Die Eigenschaften bzw. Merkmale (oder Attribute) lassen sich bei Handlungen nach verschiedenen Klassen einteilen. Die wesentlichen Merkmalsklassen, soweit sie für den Aufbau von Transaktionen interessant erscheinen, werden im nächsten Abschnitt aufgeführt.

Eine Handlung kann sich aus mehreren Handlungen zusammensetzen (komplexe Handlung). Diese "Komposition" von Handlungen sind für die Übertragung auf die Datenbank relevant, da alle Handlungen aus der Miniwelt auf "elementare" Datenbank-Operationen zurückgeführt werden müssen. In der Handlungstheorie <Gol70> wurden umfangreiche Untersuchungen unternommen, was elementare Handlungen sind und inwieweit die Komposition zur Charakterisierung eines Handlungstyps herangezogen werden kann. Für den betrachteten Bereich soll allgemein von einem Handlungskomplex dann gesprochen werden, wenn mehrere Handlungen nach bestimmten, den Komplex charakterisierenden Kriterien zusammengefaßt werden.

Ebenso wichtig wie das Ausführen einer Handlung ist das Unterlassen von Handlungen. In einer bestimmten Situation kann eine konkrete Handlung eines Handlungstyps aktualisiert oder auch (bewußt) unterlassen werden. Die Unterlassung setzt die Kenntnis des Handlungstyps und die Möglichkeit zu seiner Aktualisierung voraus. Das Unterlassen einer Handlung ist für Überlegungen zur Integrität von Interesse, denn es kann in einer bestimmten Situation gerade das Ausführen erforderlich sein.

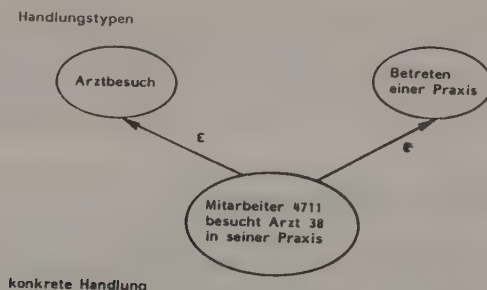
2.1.2. Handlungsmuster oder -schema

Die Zusammenfassung der einen Handlungstyp bestimmenden Merkmale wird im Rahmen dieser Arbeit als Handlungsmuster oder Handlungsschema bezeichnet. Diese Festlegung weicht geringfügig von der Literatur (<Gol70>, <KL73>, <Wri77>, <Kam80>) ab. (Der Gebrauch des Terminus "Handlungsmuster" bzw. "Handlungsschema" ist in der angegebenen Literatur nicht einheitlich.)

Eine individuelle Handlung kann bezüglich der im Handlungsschema fixierten Merkmale analysiert, beurteilt und anhand der konkreten Eigenschaften dem Handlungsschema zugeordnet werden. Ein Handlungstyp besitzt ein oder mehrere, ihn charakterisierende Handlungsschemata. Eine individuelle Handlung kann durchaus die Kriterien mehrerer Handlungsschemata erfüllen und damit unter verschiedene Handlungstypen fallen.

Abb. 2-2:

(* Beispiel konkrete Handlung fällt unter mehrere Handlungstypen *)



Hinsichtlich einer konkreten Datenbank-Anwendung in der es gilt, die Handlungen auf ein Datenbanksystem zu übertragen, ist ein Problem das Aufweisen der Handlungstypen für diese Anwendung. Im weiteren wird jedoch vielmehr prin-

ziell untersucht, inwieweit Handlungsschemata aufbereitet werden können, damit sie datenbankgerecht und als Transaktionen realisierbar sind. Aus einem Handlungsschema läßt sich ein Transaktionsschema nur dann gewinnen, wenn die zum Handlungsschema gehörenden Komponenten sich ganz oder teilweise auf die Datenbank übertragen lassen. Eine "automatisierte" Handlung zerfällt in der Regel in einen Handlungsanteil, der nachwievor in der Miniwelt stattfindet, und einen Anteil, der von dem Datenbank-Anwendungssystem übernommen wird.

Das skizzierte Handlungsschema geht in ein "Transaktionsschema" über und wird als Transaktionsprogramm realisiert. Die einzelnen Komponenten eines Handlungsschema sind dabei auf ihre Übertragbarkeit zu untersuchen und geeignete Mittel zur Darstellung anzugeben. Die in einem Datenbanksystem vorhandenen Möglichkeiten sind dabei zu berücksichtigen.

Um eine konkrete Handlung h einem bestimmten Handlungstyp H zuordnen zu können, gilt es die Merkmalsklassen M_i oder Komponenten eines dem Typ H zugeordneten Handlungsschemas $H\langle M_1, \dots, M_n \rangle$ zu betrachten.

Diese Klassifikation von Handlungsmerkmalen geht im wesentlichen auf Rescher <Res67> zurück und wird im folgenden durch Aspekte von Goldmann <Gol70>, Kamlah <KL73>, und Wright <Wri77> ergänzt.

(1) Subjektklasse einer Handlung

Jede konkrete Handlung wird von einem Subjekt (Agenten) ausgeführt bzw. initiiert. Ein wesentliches Kriterium für die Entscheidung, ob eine Handlung unter einen Handlungstyp fällt, ist die Zugehörigkeit des ausführenden Subjekts zur Subjektklasse des Handlungstyps.

Ein einfaches Beispiel für die Subjektabhängigkeit von Handlungen:

Für das Beispiel des Außendienst (siehe Anhang) ist ein "Arztbesuch" nur dann erfolgt, wenn der Arzt von einem Außendienstmitarbeiter besucht wurde. Ein Besuch durch irgendeine andere Personen erfüllt nicht das Handlungskriterium.

Die Subjektklasse einer Handlung umfaßt alle jene Subjekte, die befähigt sind, die Handlung zu aktualisieren. Die Befähigung ist dabei deutlich von der Zulässigkeit abzugrenzen. Die Zulässigkeit orientiert sich an den deontischen Kategorien des "Erlaubten", "Gebotenen" und "Verbotenen". Die Befähigung gibt die rein potentielle Möglichkeit zur Ausführung an. Es besteht dennoch ein unmittelbarer Zusammenhang, denn ein Subjekt muß zu einer Handlung (bzw. zu einer Transaktion) befähigt sein, wenn sie für das Subjekt geboten oder erlaubt ist. Andererseits wird man in einer Datenbank-Anwendung versuchen, die Befähigung eines Subjekts möglichst auf für ihn erlaubte und gebotene Transaktionen zu beschränken. Der deontische Gesichtspunkt einer Handlung bzw. Transaktion wird in Kapitel 5 im Rahmen der Integrität behandelt, der zusätzliche personengebundene Aspekt wird in Kapitel 6 noch ergänzt.

(2) Objektklasse einer Handlung

Eine Handlung findet auf oder mit Objekten statt. Es sind dabei die unmittelbar betroffenen Objekte, die durch eine Handlung erzeugt, verändert oder verbraucht werden, von den mittelbar betroffenen Objekten, die nur der Handlung als Mittel dienen oder deren Eigenschaften den Ablauf der Handlung beeinflussen, zu unterscheiden.

Für ein Handlungsschema ergibt sich eine Objektklasse, die in zwei disjunkte Teilklassen, der unmittelbar betroffenen und der mittelbar betroffenen Objekte, zerfällt.

Handlungen sind nur dann in die Form von Transaktionen übertragbar, wenn ihre Handlungsobjekte in der Datenbank ganz oder zumindest teilweise repräsentiert sind. Für das Datenbank-Anwendungssystem bedeutet dies, daß die Objektklassen in Form von (abgeleiteten) Relationen und die Objekte in der Form von Tupeln zur Verfügung stehen. (Die relationale Darstellung bedeutet nicht, daß den Benutzern als Schnittstelle die Relationen sichtbar sein müssen. In der Form von Masken können z.B. durchaus die gewohnten Formulare wieder zur Verfügung gestellt werden.) Im Anhang 1 wird am Beispiel eines pharmazeutischen Außendienstes aufgezeigt, wie Objekte der Miniwelt durch Tupel der Datenbank repräsentiert werden können.

(3) Handlungssituation

Eine Handlung findet in einem gegebenen Zustand der Miniwelt statt. Eine Handlung läßt sich charakterisieren durch die Zustände vor, während und nach ihrer Ausführung. Die Festlegung dieser für eine Handlung typischen Situationen heißen analog zu Wright (<Wri77>, S.87) Handlungssituationen. Die eine Handlung charakterisierenden Situationen lassen sich durch die Angabe von Prädikaten über die Objektklasse der Handlung beschreiben.

Es besteht eine direkte Analogie zu den in der Programmierung <Gri82> üblichen Vor- und Nachbedingungen von Operationen. Die Voraussetzung zur Ausführung einer Handlung entspricht den Vorbedingungen, die Beschreibung des Resultats der Handlung den Nachbedingungen aus der Programmierung. Beispiele zur Übertragung von Handlungen in die Form von Transaktionen finden sich im Anhang 3 und 4.

Für die Übertragung der Handlung in die Form von Transaktionen ist zu bemerken, daß nur jene Prädikate prüfbar sind, die sich auf Objekte in der Datenbank beziehen. Hierbei ist zu unterscheiden, in welchem Zustand sich die Objekte der Datenbank (Tupeln der Relationen) befinden und welche Situation in der Miniwelt vorliegt. Die Ausführbarkeit einer Transaktion hängt von beiden Zuständen ab. Die Kenntnis des Zustands der Miniwelt beschränkt sich für die Transaktion nur auf die übergebenen Parameter. Nur diese sind prüfbar.

Die Angabe der Objektklasse und die Prädikate zur Beschreibung des Übergangs gehen direkt in die Spezifikation der Transaktionen ein (vgl. Abs. 2.2 und Kap. 3, 4). Aus den bisherigen Betrachtungen erkennt man, daß der Subjektbezug in der Spezifikation von Transaktionen vernachlässigt wurde, obwohl er bei Handlungen gegeben ist.

(4) Verlaufs- oder Ablaufschema einer Handlung

In der Handlungstheorie ist zur Charakterisierung einer Handlung auch die Art und Weise des Verlaufs Bestandteil des Handlungsmusters. Eine Handlung kann demnach aktualisiert werden durch die Ausführung einer Folge von (Teil-)Handlungen. Der Aufbau dieser Folgen bildet ein weiteres Merkmal eines Handlungstyps. Es gelten nur bestimmte Folgen von (Teil-)Handlungen als typspezifisch.

Die Angabe der Handlungsabläufe in Teilhandlungen ist nicht problemlos (vgl. <Gol70>), da sich die Frage nach "elementaren" oder atomaren Handlungen stellt, d.h. nach der Tiefe und Granularität der Verfeinerung. Eine Handlung kann eine Menge für sie typischer Handlungsfolgen besitzen.

Für die transaktionsorientierte Betrachtungsweise ist dies die Frage nach dem Algorithmus, der nötig ist, um die Spezifikation der Transaktion (festgelegt durch die Objekte

und die Vor- und Nachbedingungen) zu implementieren. Für die Übertragung der Teilschemata des Handlungsmusters in programmsprachliche Anweisungen ist zu bemerken, daß es gerade ein Vorteil der Automatisierung ist, nicht die der Miniwelt vorhandenen Muster zu übernehmen, sondern durch andere maschinenspezifische, in diesem Fall datenbankadäquate Operationen zu ersetzen.

Ein manuelles Handlungsablaufschaema kann nur als Leitlinie für den Implementationsteil einer Transaktion dienen. Es ist wichtig, logische Abhängigkeiten von Teilhandlungen in einem komplexen Handlungsschema zu erkennen und diese im Transaktionsprogramm zu berücksichtigen.

Inwieweit eine Handlung zur Automatisierung in Teilhandlungen zu zergliedern ist, ist für eine Datenbank-Anwendungssystem pragmatisch zu beantworten:

- 1.) möglichst grob,
da sich Teilhandlungen wieder wie Handlungen behandeln und damit bei Bedarf auch weiter zergliedern lassen,
- 2.) nicht feiner,
als es bereits automatisierte Handlungen sind, \n(d.h. die Operationen des Datenmodells als virtuelle Maschine bilden eine untere Schranke).

Zusammenfassende Bemerkungen:

In den weiteren Abschnitten, in denen der Übergang von den Handlungsstrukturen zu den Transaktionsstrukturen von Bedeutung ist, werden die jeweiligen Komponenten aufgegriffen und explizit behandelt. Nach dieser Skizzierung der Grundkomponenten eines Handlungsschemas gilt zusammenfassend:

- (1) Eine Handlung oder Handlungstyp ist charakterisiert durch die für ihn spezifischen Merkmale, die sich in Handlungsschemata festlegen lassen. Ein Handlungstyp kann durch mehrere Handlungsschemata charakterisiert werden.
- (2) Ein Handlungsschema gliedert sich in:
- Subjektklasse,
(Menge der Subjekte, die zur Handlung fähig sind);
 - Objektklasse,
(Menge der Objekte, die von der Handlung betroffen sind);
 - Handlungssituationen,
(die Voraussetzungen für die Ausführung und die Beschreibung der Wirkung einer Handlung);
 - Verlaufsschema,
(die Folge von Handlungen, in die sich eine Handlung zergliedern läßt).
- (3) Eine konkrete Handlung fällt unter einen Typ, wenn sie die Eigenschaften besitzt, die in einem den Typ charakterisierenden Handlungsschema festgelegt sind. Eine konkrete Handlung kann die Eigenschaften zur Erfüllung mehrerer Schemata besitzen und folglich auch unter mehrere Handlungstypen fallen.

Abb. 2-3:

(* Zusammenhang Typ, Schema, Ausprägung *)

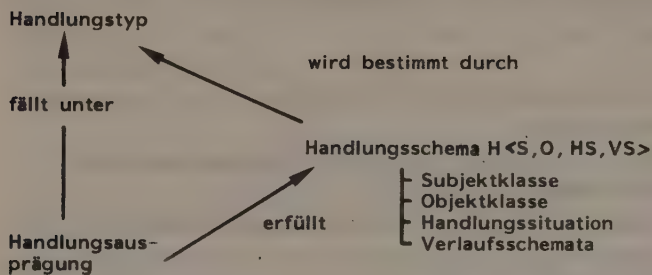


Abbildung 2-4 illustriert, wie die Komponenten eines Handlungsschema in Komponenten einer Transaktion" übergehen:

Abb. 2-4:

(* Übergang Handlung - Transaktion *)

+-----+-----+		
	Handlungsschema -->	"Transaktionsschema"
+-----+-----+		
	Objektklasse -->	Objektklasse
		Relationen der
		Datenbank
		Eingabe - Parameter
		Ausgabe - Parameter
	Handlungssituation -->	Vor- und
		Nachbedingungen
	Verlaufsschema -->	Algorithmus
	Subjektklasse -->	Subjektklasse
		(eingeeengt durch
		deontische Kriterien)
	Ergänzt durch	
	Handlungskenn-	Transaktions-
	zeichnung -->	kennzeichnung
		(zur eindeutigen
		Identifikation)
+-----+-----+		

2.1.3. Begründung und Rechtfertigung von Handlungen

Die Problematik der Übertragung von Handlungsstrukturen auf Transaktionsstrukturen erstreckt sich nicht nur auf die Bereitstellung von Transaktionen, sondern beinhaltet auch deren Überwachung.

Der in der Handlungstheorie behandelte Aspekt der Handlungsbegründung und damit verbunden der deontische Aspekt, ob in einer gegebenen Situation eine Handlung erlaubt, geboten oder verboten ist, ist im Aufbau einer Transaktionskontrolle zu berücksichtigen. Hierzu müssen die Kriterien angebbbar sein, nach denen entschieden werden kann, warum in einer vorliegenden Situation so und nicht anders zu handeln ist.

Nach intentionalistischer Auffassung ist bei der Ausführung einer Handlung immer beabsichtigt eine gewisse Wirkung (Zweck) zu erzielen. Eine konkrete Handlung ist deshalb zu beurteilen, inwieweit sie ein geeignetes Mittel darstellt, den gesetzten Zweck zu erreichen. Als Zweck ist die zu erreichende Situation zu verstehen (vgl. (vgl. <Schw76>, S.130, <Wed80>). Der Begründung oder Rechtfertigung einer Handlung liegt damit eine teleologische Beziehung zugrunde, die ausdrückt, welchem Zweck (Ziel) die Ausführung einer Handlung als Mittel dient (vgl. <WW79>, S.135ff).

Das Setzen von Zwecken, d.h. die Vorgabe, in einer Situation einen bestimmten Zweck zu verfolgen, ist im Bereich der Kulturwissenschaft Gegenstand intensiver Diskussion (vgl. <Schw76>, S.199). Es sind insbesondere die einzelnen Ziele einer Anwendung zu einander in Beziehung zu setzen und eine Wertestruktur zu erstellen (vgl. <WW79>, S.148ff). In der betrachteten Miniwelt einer Datenbank-Anwendung kann diese Problematik stark reduziert werden. Man kann davon ausgehen, daß die Zwecksetzungen in der Regel geklärt bzw. auf einfache Weise klärbar und bewertbar sind.

In den Kulturwissenschaften (<Schw76>, S.28) bezeichnet man Aufforderungen in bestimmten Situationen Zwecke zu verfolgen als Normen, die universell (d.h. für alle Handelnden) und generell (d.h. zu jeder Zeit) gelten. In der Miniwelt einer Anwendung kann man nicht von universellen Zwecken bzw. Normen ausgehen, sondern die Zwecke werden in der Regel zeit- und personenabhängig festgelegt sein. Diese Aufforderungen zur Verfolgung von nicht universalisierten und nicht generalisierten Zwecken heißen Vorschriften.

Die Ausführung einer Handlung ist gerechtfertigt, wenn sie mit den Vorschriften der Miniwelt verträglich ist. Es ist der Grund für die Ausführung oder Unterlassung einer Handlung zu finden und mit den Vorschriften in Beziehung zu setzen.

Als Gründe einer Handlung können dienen:

- die Handlungsursache,
d.h., es lag eine Situation vor, die die Ausführung einer Handlung erforderlich machte; es existiert eine entsprechende Vorschrift oder Norm zur Ausführung oder Unterlassung der Handlung;
- das Handlungsziel,
d.h., die durch die Handlung erreichte Wirkung entspricht dem gesetzten Zweck oder sie ist Teil einer Handlung, die den Zweck erreichen wird;
- die Handlungsintention,
d.h., die zu erreichen beabsichtigte Situation entspricht dem gesetzten Zweck oder er kann durch Ausführung weiterer Handlungen erreicht werden.

Zur Begründung von Handlungen kann man auf die Intention auch dann zurückzugreifen, wenn die Ausführung gescheitert ist. Die intendierte Situation nicht erreicht wurde. Die Begründung von Handlungen ist unter deontischen Aspekten wichtig, wenn zu entscheiden ist, inwieweit eine Handlung erlaubt, geboten oder verboten ist. Es wird dann die Begrün-

dung auf ihre Rechtfertigung hin geprüft.

In einem Datenbank-Anwendungssystem ist zu entscheiden, ob eine Transaktion gerechtfertigt ist. Das Anwendungssystem hat dies zu überwachen. Ist die Transaktionsausführung mit den Zwecksetzungen bzw. Vorschriften der Anwendung verträglich, ist sie erlaubt, ist sie in der gegebenen Situation unabdingbar für den verfolgten Zweck, ist sie geboten, ist sie mit den Zielen nicht vereinbar, ist sie verboten. Die Überprüfung kann nur erfolgen, wenn die Vorschriften bzw. Zwecke explizit fixiert sind.

Die Prüfung zu Beginn einer Transaktion bedeutet, daß die Wirkung der Transaktion bekannt sein muß. (Dieser Gesichtspunkt wird in Kapitel 4 und 5 noch weiter behandelt). Ist die Wirkung der Transaktion apriori durch die Angabe von Nachbedingungen bestimmbar, läßt sich die Zulässigkeit ihrer Ausführung apriori entscheiden. Bezieht man das Subjekt in die Entscheidung mit ein, so gelangt man zu einem personenbezogenen Schutzkonzept (vgl. Kap. 6).

Ist eine Prüfung der Transaktion apriori nicht möglich, so ist dennoch ihre Wirkung auf der Datenbank und auf die Belegung der Ausgabeparameter nach der Ausführung prüfbar. Ein nicht zu akzeptierendes, den gesetzten Zwecken widersprechendes Resultat ist durch das Transaktionskontrollsystem rückgängig zu machen.

Ist die erzielte Wirkung einer Transaktion nur ein Schritt hin auf einen beabsichtigten Zweck, so ist es notwendig, zur Erreichung des Ziels weitere Transaktionen einzuleiten. Auch diese Aufgabe kann das Kontrollsystem ebenfalls unterstützen.

Eine Grenze für das Übertragen von Handlungsschemata in eine prozedurale Form und die Prüfung der Handlungsbegründung zu Beginn einer Transaktion ergibt sich, wenn der Hand-

lungsablauf nicht determiniert ist. Dies ist z.B. der Fall, wenn menschliche Entscheidungen während der Handlungsausführung zu berücksichtigen sind.

Für ein umfassendes Datenbank-Anwendungssystem ist ein Kontrollsystem notwendig, das nicht nur die deontischen Vorbedingungen der Transaktionen prüft, sondern auch die Gültigkeit der Nachbedingungen überwacht und bei Bedarf zusätzlich weitere Transaktionen einleitet. Eine ausführliche Behandlung hierzu findet sich in den Kapiteln 4 und 5.

2.1.4. Die Gray'schen Kriterien

Die Gray'schen Kriterien (vgl. Abs. 1.2.2) grenzen die Menge der automatisierbaren Handlungen normativ ein. Es lassen sich ausschließlich Handlungen in Transaktionen transformieren, die jene Anforderungen der Atomarität, Konsistenz, Isolation*) und Dauerhaftigkeit auch in der Miniwelt erfüllen.

Eine Transaktion wird atomar (im Gray'schen Sinne), wenn die zugehörige Handlung in der Miniwelt ebenfalls eine atomare Form hat, d.h., ist die Handlung einmal eingeleitet, so ist sie auch in der Miniwelt zu Ende zu führen. Bei einer Unterbrechung der Ausführung wird sie auch in der Miniwelt zurückgesetzt, das heißt der Anfangszustand ist wieder gültig. Es ist nicht zu verkennen, daß gerade hierein eine der Schwierigkeiten des Entwurfs von Transaktionen liegt.

Die konsistenzzerhaltende Eigenschaft der Transaktionen

*) (Die Isolation ist in den ursprünglichen Gray'schen Kriterien nur implizit enthalten. Härder/Reuter <HR83> führen explizit als Transaktionskriterien auf "Atomicity", "Consistency", "Isolation", "Durability".)

(im Gray'schen Sinne) ist in Bezug auf die Handlungen ebenso vielfältig wie für die Datenbank zu interpretieren. Es gibt eine technische Integrität, die die korrekte Handlungsweise (die korrekte Durchführung einer Handlung) betrifft. Es existiert eine deontische Integrität, die die Erzeugung nur solcher Zustände zuläßt, die den Vorschriften und Normen der Miniwelt entsprechen. Diese Beschränkung erscheint auch in einem Datenbank-Anwendungssystem sinnvoll zu sein. Sie findet in Kapitel 5 Eingang in der Überwachung der semantischen und der deontischen Integrität.

Die vollständige Isolation der Transaktionen entspricht der vollständigen Isolation bei Handlungen. Es ist somit nicht möglich Handlungen in ein Datenbank-Anwendungssystem zu übertragen, die innerhalb ihrer, als atomar geltenden Abläufe miteinander in Wechselwirkung treten. Dies ist eine echte Einschränkung für die Realisierung von Handlungen der Form von Transaktionen.

Das Kriterium der Dauerhaftigkeit der Wirkung von Handlungen hat zwei Aspekte. Was einmal in der Miniwelt geschehen ist, kann nur durch weitere Handlungen wieder rückgängig gemacht werden. Zum anderen beinhaltet die Forderung nach Dauerhaftigkeit auch, daß die Wirkung einer Handlung evident wird. Nachfolgende Handlungen nehmen auf den erzeugten und nicht auf einen zurückliegenden Zustand Bezug. Die Wirkung einer Handlung also als "verbindlich" (committed) für das weitere Handeln anerkannt wird.

Für die Übertragung von Handlungen werden in dieser Arbeit die Gray'schen Kriterien zugrunde gelegt.

2.2. Zur Struktur von Transaktionen

Die Schwierigkeiten bezüglich der Spezifikation und Implementierung von Transaktionen, die sich aus der Übertragung von Handlungen in ein Datenbank-Anwendungssystem ergeben, werden im weiteren dieses Kapitels untersucht. Die Struktur einer Transaktion wird zuerst isoliert betrachtet, danach werden in Kap. 3 die Aspekte zur Integration der Transaktionen zu einem Gesamtsystem behandelt. Die gängige Vorgehensweise für die Umsetzung von Handlungen in Transaktionen (vgl. <ALM82>, <Wed79>, <RS81>) ist sicherlich die Erstellung spezifischer Anwendungsprogramme, die jeweils die gewünschte Anwendungsoperation auf der Datenbank als Transaktion realisieren. Im Bereich der semantischen Datenmodellierung wurde diese Zuordnung beibehalten und die Spezifikation von Transaktionen auf diesem Ansatz aufgebaut <Bro81>. Im ersten der folgenden Abschnitte wird diese Vorgehensweise näher untersucht. Die Gründe, weshalb dieses Vorgehen unzureichend ist, und erste Ansätze für die notwendigen Ergänzungen sind Inhalt der weiteren Abschnitte. Die Idee des Transaktionsprogramms kommt letztlich aus der Stapelverarbeitung <Gra77>. Es wird dabei ein funktionaler determinierter Übergang von einem Datenbankzustand in einen anderen zugrunde gelegt.

Transaktionen sind als anwendungsspezifische Operationen durch schrittweise Konstruktion aus den elementaren Datenbank-Operationen zu erzeugen. Innerhalb einer Transaktion entstehen so anwendungsspezifische Aktionen, die bereits sehr komplexe Teilaufgaben realisieren können. In diesem Zusammenhang ist auf die Restriktionen hinzuweisen, die es verhindern, daß elementare Datenbank-Operationen zu beliebig komplexen Operationen komponiert werden können. Komplexe Operationen, die alle Transaktionseigenschaften erfüllen, bilden letztlich Transaktionsprogramme. Die Transaktion bildet daneben einen Prozeß auf einer, dem Benutzer zur Verfügung stehenden virtuellen Maschine. Der datenbankspezi-

fische Aufbau dieser virtuellen Maschine wird in Kap. 3 erörtert.

2.2.1. Ein bekannter Ansatz - die Transaktionsprozedur

Die Transaktionsprozedur ist ein erster, in der Literatur bekannter Ansatz die Vorgänge der Anwendung in Transaktionen abzubilden. Diese Vorgehensweise wird z.B. von Reimer/Schmidt <RS81>, Brodie <Bro81> und anderen <Atk78>, <Web78>, <RZ82> vorgeschlagen. Ausgehend von der Funktionalität der Transaktion wird dieses Konzept im weiteren exemplarisch vorgestellt. Die wichtige, konsequente Trennung zwischen Spezifikation und Implementation von Operationen (hier Transaktionen auf der Datenbank) wird ebenfalls verdeutlicht.

Die Transaktion stellt (<PP77>, <Gra80>) einen funktionalen Übergang von einem integeren (konsistenten) Datenbankzustand in einen wiederum konsistenten Zustand (vgl. auch Abs. 1.2.1) dar. Der Übergang ist auf die Ein- und Ausgabeparameter zu erweitern, wenn man die Transaktion als eine Prozedur auffaßt.

T : DATABASE X INPUT ----> DATABASE X OUTPUT

In Hoare'scher Notation mit entsprechenden Bedingungen für den Übergang (vgl. <Gra80>), mit der sich auch nicht funktionale Übergänge beschreiben lassen (vgl. Abs. 4.3), kann man schreiben:

<pre-condition> < T > <post-condition>

Entsprechend zu Gray <Gra80> hat man zwischen transaktionsinvarianten Bedingungen und den Bedingungen, die den eigentlichen funktionalen Übergang spezifizieren, zu diffe-

renzieren.

```
INT(db-vorher) AND <variant-precondition>  
< T >  
INT(db-nachher) AND <variant-postcondition>
```

Die transaktionsinvarianten Bedingungen entsprechen den semantischen und strukturellen Datenintegritätsbedingungen (vgl. dazu Abs. 5.3).

verschiedene Ansätze

Im folgenden werden die Ansätze von Reimer/Schmidt <RS81> und Brodie <Bro81> exemplarisch betrachtet. Reimer/Schmidt fassen die Transaktionen als Prozeduren auf. (Sie verwenden dabei als Programmiersprache und als Datenmanipulationssprache PASCAL/R. PASCAL/R steht sowohl für eine Erweiterung von PASCAL um Konstrukte zur Handhabung von Relationen im Sinne des Relationalen Datenmodells als auch für ein Datenbanksystem <Schm80>.)

Brodie dagegen entwickelt eine weitergehende Vorgehensweise, indem er erstens konsequent zwischen Spezifikation und Implementierung unterscheidet und zweitens sogenannte Aktionsprozeduren einführt. Eine Transaktion greift über diese Prozeduren auf die Datenbank zu.

Reimer/Schmidt verwenden (im Gegensatz zu Brodie) den Begriff der Transaktionen synonym zu "Aktionsprozedur" (<RS81>, S.7). Dieser Prozedur werden neben den Parameterwerten, die aus der Miniwelt versorgt werden, auch die Relationen der Datenbank als Eingabe- bzw. Ausgabeparameter übergeben. Die übergebenen Relationen können abgeleitet, d.h. durch eine Abfrage auf der Datenbank eingeeengt sein.

Die Notation für eine Transaktion sieht nach Reimer/Schmidt <RS81>, wie folgt, aus:

```
TRANSACTION t-name (<parameter-list>);
```

```
    <declaration-part>
```

```
BEGIN
```

```
    :
```

```
    <statements>
```

```
    :
```

```
END;
```

Entscheidend ist die Übergabe relationaler Parameter an die Transaktionsprozedur. Im aufrufenden Hauptprogramm sind diese zu definieren. (Eine ausführliche Darstellung findet sich in Reimer/Schmidt <RS81>).

Ein stark vereinfachtes Beispiel auf der im Anhang 1 beschriebenen Pharma-Datenbank ist der "Eintrag eines Arztbesuchs":

```
TRANSACTION ArztBesuchEintragen
```

```
    (IN      Eintrag:      ...;
```

```
     INOUT  ARZTBESUCH:    ...;
```

```
     IN     ARZT:          ...;
```

```
     IN     MITARBEITER:   ...;)
```

```
    (* Der Eintrag ist bereits als Tupel aufgebaut *)
```

```
BEGIN
```

```
IF NOT SOME a IN ARZTBESUCH AND NOT SOME n IN Eintrag
```

```
    (a.ArztNr=n.ArztNr AND a.MA-Nr=n.MA-Nr
```

```
    AND a.DATUM=n.date)
```

```
    THEN ARZTBESUCH :+ Eintrag
```

```
END;
```

(Das Beispiel "Eintrag eines Arztbesuchs" wurde auf den eigentlichen Eintrag reduziert. Korrekterweise sind verschiedene Voraussetzungen, wie z.B. die Zuständigkeit des Mitarbeiters, zu prüfen. Die ausführliche Darstellung des Beispiels findet sich unter Berücksichtigung der in der Arbeit erbrachten Ergebnisse im Anhang 2.)

Die "Transaktionsprozeduren" unterscheiden sich von normalen Prozeduren zum einen durch die Forderung, daß die Atomarität der Prozedurausführung zu gewährleisten ist, zum anderen, daß der Prozedur neben den üblichen Parametern auch Relationen übergeben werden.

Diese Vorgehensweise widerspricht dem Prinzip der "Lokalität", nämlich der Trennung zwischen Datenbereitstellung und Verarbeitung <Wed79>. Die Bereitstellung von Objekten der Datenbank sollte demnach innerhalb der Prozeduren erfolgen. Ein weiterer Mangel dieses Ansatzes ist das Fehlen einer klaren Trennung zwischen Spezifikation und Implementation.

Zur Spezifikation von Transaktionsprozeduren

Brodie <Bro81> vollzieht in seinem Ansatz, der im wesentlichen dem Prozedurgedanken entspricht, die Trennung zwischen Spezifikation und Implementation. Er integriert auch die Datenbereitstellung in die Transaktionsprozeduren, in der Form von "Aktionen". Der Aufbau der Aktionen wird im folgenden Abschnitt 2.2.2 betrachtet.

Die Spezifikation einer Transaktion ist die (formale) Beschreibung des funktionalen Übergangs, z.B. durch die Angabe von Vor- und Nachbedingungen. Dabei wird vom Algorithmus und seiner programmtechnischen Realisierung abstrahiert.

Die Spezifikation einer Transaktion besteht nach Brodie aus den folgenden Komponenten:

- dem Namen der Transaktion;
- der Parameterliste (aus Ein- und Ausgabeparametern);
- den Relationen, auf die lesend zugegriffen wird (direkt oder durch benutzte Aktionsprozeduren);
- den Relationen, auf die ändernd zugegriffen wird (direkt oder durch benutzte Aktionsprozeduren);
- lokalen Variablen der Transaktion

Die Wirkung der Transaktion wird spezifiziert durch

- die Vorbedingung und
- die Nachbedingung.

Ergänzt wird die Spezifikation (bei Aktionen) durch eine Liste der verwendeten

- Datenbank-Operationen.

In Anlehnung an die Vorgehensweise von Brodie <Bro81> kann man zur Spezifikation des funktionalen Übergangs entsprechende Sprachkonstrukte einführen. Die Ähnlichkeit zu der in der Systemprogrammierung üblichen Vorgehensweise (wie z.B. in <SWL77>, <Ker82>, <Gri82>) ist augenfällig, der Bezug wird in Kap. 3 und 4 noch vertieft.

Eine Transaktion läßt sich nach Brodie in folgender Form spezifizieren:

```
TRANSACTION      transaction-name
                  (<input-parameter-list>,
                   <output-parameter-list>)
SCOPE            (<database-relations>,
                  <local-variables>)
PRE-CONDITION:   <pre-condition>
POST-CONDITION:  <post-condition>
```

Im "Scope" werden die Variablen vereinbart und die benötigten Relationen der Datenbank angegeben. Die Vor- und Nachbedingungen, informal als boolsche Funktionen notiert, umfassen sowohl transaktionsspezifische wie auch datenbank-invariante Bedingungen. Brodie nimmt diesbezüglich keine Differenzierung der Bedingungen vor. Die Angabe der invarianten Bedingungen bei jeder Transaktion ist müßig. Es genügt zu fordern, daß die transaktionsspezifischen Bedingungen mit den invarianten Bedingungen verträglich sind, denn invarianten Bedingungen tragen nichts zur Spezifikation des Übergangs bei.

Ein einfaches Beispiel für eine Spezifikation einer Transaktionsprozedur nach dem Ansatz von Brodie (wiederum am "Eintrag eines Arztbesuchs") ist:

```
TRANSACTION      ArztBesuchEintragen;  
                  (i-mitarbeiternr; i-arztnr; i-datum;  
                   i-folgenr; i-bemerkung; o-korrekt );  
SCOPE            ARZTBESUCH;  
                  ....  
PRE-CONDITION:   arzt-existiert (i-arztnr);  
                  mitarbeiter-existiert (i-mitarbeiternr);  
                  ....  
POST-CONDITION:  arztbesuch-existiert  
                  (i-arztnr, i-mitarbeiternr, i-datum);  
                  ....
```

(Die "...." stehen für die Angabe weiterer Bedingungen.)
(Wie erwähnt findet sich eine ausführliche Spezifikation und Implementation des entsprechenden Beispiels im Anhang).
Die Angabe der invarianten Bedingungen der Datenbank kann man auf die limitieren, die die im "Scope" angegebenen Relationen betreffen.

Zur Implementation von Transaktionsprozeduren

In der Implementation einer Transaktion wird in einer Programmier- und Datenmanipulationssprache ein Algorithmus angegeben, der die Übergangsbedingungen der zugehörigen Spezifikation erfüllt. Bei der Spezifikation handelt es sich um eine Abstraktion von der Implementation der Transaktion. Brodie <Bro81> gibt nur eine sehr oberflächliche Struktur für die Implementation von Transaktionen an:

TRANSACTION

SCOPE

```
IF      <pre-condition>
THEN    ....
        <action>
        ....
CHECK   <post-condition>;
```

Der Ausdruck <action> steht für den Aufruf von Aktionsprozeduren (vgl. Abs. 2.2.2), die auf den Anweisungen der Datenbankmanipulationssprache aufsetzen. Die Vorbedingung wird durch die einleitende IF-Anweisung abgefangen. Die Fehlerreaktion ist implizit. Sie bewirkt, daß die Transaktion nicht ausgeführt wird. Die abschließende CHECK-Anweisung in der Implementation bedeutet, bei Nichterfüllung erfolgt ein Rücksetzen der Transaktion. Eigentlich ist die Überprüfung überflüssig, wenn das Datenbanksystem die Isolation der Transaktionen gewährleistet und die Transaktion korrekt implementiert ist.

Aufgrund der Gray'schen Kriterien ist die Ausführung einer Transaktionsprozedur für den Benutzer atomar. Der Benutzer übergibt beim Aufruf die Eingabeparameter und erhält nach Beendigung der Transaktion ein entsprechendes Resultat zurück. Damit ist im Prinzip das Problem der Transaktions-spezifikation und -implementation auf gängige Methoden der Programmierung zurückgeführt. Es scheint, auf dieser Auffassung beruht sowohl der Ansatz von Brodie <Bro81> als auch der von Reimer/Schmidt <RS81>.

Grenzen des Ansatzes der Transaktionsprozedur

Bei der dargestellten Vorgehensweise des "Prozeduransatzes" treten einige Mängel auf, die im folgenden kurz verdeutlicht werden.

Es bleibt u.a. unberücksichtigt, daß neben dem Zustand der Datenbank auch der externe Zustand der Miniwelt den Verlauf einer Transaktion bestimmen kann. An dem vorausgegangenem Beispiel "Eintrag eines Arztbesuchs" läßt sich dies belegen. In der angegebenen Transaktionsprozedur wurde als Parameter die Arzt-Nummer (zum Attribut ARZT.ArztNr) übergeben. Dies war notwendig, um eine eindeutige Zuordnung eines Arztes aus der Datenbank zum Besuchseintrag zu gewährleisten. In der Praxis wäre für den Außendienstmitarbeiter die Eingabe des Namens des Arztes angenehmer. Dies führt u.U. zur Qualifikation mehrerer, in der Datenbank gespeicherter Ärzte. Die Eindeutigkeit kann durch den eintragenden Außendienstmitarbeiter unverzüglich erreicht werden, wenn man ihm die qualifizierten Ärzte anzeigt und eine Auswahl des gewünschten Arztes ermöglicht. Diese Art der Abarbeitung der Operation "Eintrag eines Arztbesuchs" bedeutet jedoch ein Verlassen des Gedankens der Realisierung von Transaktionen in der dargestellten prozeduralen Form. Durch den jetzt zusätzlich erforderlichen Dialog mit dem Benutzer ist zum Zeitpunkt des Beginns der Transaktion kein funktionaler Übergang mehr gegeben. Dieses einfache Beispiel mag künstlich konstruiert erscheinen. Weitere, praxisnähere Beispiele (vgl. dazu Abs. 4.3) können durchaus angeführt werden.

Eine Erweiterung der Problematik des Prozeduransatzes ergibt sich aus dem Wechselspiel zwischen Benutzer und Datenbank. Die Ausführung der Operation "Planung von Arztbesuchen" ist hierfür ein Beispiel. Dem Außendienstmitarbeiter wird eine Menge von Alternativen für den Besuch geeigneter Ärzte vorgegeben. Aus diesen Möglichkeiten trifft der Mitarbeiter eine geeignete Auswahl aufgrund von Kriterien,

die sich aktuell auf eine in der Miniwelt vorliegende Situation beziehen.

Es gibt erste Ansätze für Datenbank Anwendungen, die eine Integration der Dialogentscheidungen in das Transaktionsprogramm vorsehen. Eine Möglichkeit wird z.B. in DIAL <HB80> aufgezeigt. Davies <Dav78> weist durch die Möglichkeit des "inline processing" auf die Notwendigkeit hin, daß der Benutzer selbst Teil der Transaktionsausführung wird. Weitere Ansätze finden sich hierzu auch in <Wed83>.

Eine weitere Unzulänglichkeit des vorgestellten Ansatzes besteht in dem Art und Weise der Sichtdefinition. Es bestehen keine <Bro81> oder nur eingeschränkte <RS81> Möglichkeiten Definition einer indirekten Zuordnung zwischen Variablen im Anwendungsprogramm und der Datenbank. Für den Aufbau eines Anwendungssystems ist es aber notwendig, die Repräsentationsfunktion von Sichten (vgl. Abs. 3.1) explizit miteinzubeziehen, um die Datenunabhängigkeit zu gewährleisten (vgl. Abs. 1.1.2).

Diese Punkte verdeutlichen, daß die Bildung von Transaktionen in der Form von Prozeduren, wie sie z.B. Brodie u.a. vorgeschlagen haben, nicht hinreichend sind für die Erstellung eines umfassenden, transaktionsorientierten Anwendungssystems. In einem neueren Ansatz von Rebsamen et al. (<RRUZ82>, <RZ82>) wird bereits zwischen standardisierten (die den Transaktionsprozeduren entsprechen) und freien Transaktionen (die auf dem Dialog in einer Abfragesprache beruhen) unterschieden. Auch damit wird gezeigt, daß durch Transaktionsprozeduren, wie in Kapitel 4 noch ausführlich behandelt wird, nur ein Teil der Handlungen einer Miniwelt realisierbar sind.

Die angegebenen Kritikpunkte zeigen aber auch, daß beim Zergliedern der Handlungen in Teilhandlungen, diese trotzdem im Sinne der Transaktion zusammengefaßt, einen atomaren und

konsistenten Übergang für den Benutzer und die Datenbank ergeben müssen. Das Datenbank-Anwendungssystem hat diesen Übergang zu überwachen.

2.2.2. Von universellen Datenbank-Operationen zu anwendungsspezifischen Operationen

Die Begriffe oder Objekttypen der Miniwelt werden durch die Relationen der Datenbank repräsentiert, wie eingangs (vgl. Abs. 1.1.1) ausgeführt wurde. Die zu automatisierenden Handlungen aus der Miniwelt bestehen (vgl. Abs. 2.1.2) aus einer Folge von Teilhandlungen, die zu Handlungen nach den Gray'schen Kriterien zusammenzufassen sind. Die Realisierung der bezüglich der Datenbank elementaren Teilhandlungen erfolgt durch das Ausführen mengenorientierter Datenbank-Operationen.

Geht man von den elementaren Datenbank-Operationen (z.B. in QUEL) aus (vgl. Abs. 1.2.1), so sind diese generell auf alle Relationen anwendbar. Zur Realisierung einer elementaren anwendungsspezifischen Teilhandlung sind sie nun an spezifische Relationen zu binden. Die gebundenen Datenbank-Operationen lassen sich als elementare "Aktionsprozeduren" realisieren. Dies wurde von Brodie <Bro81> bereits skizziert. Beim weiteren Aufbau eines Datenbank-Anwendungssystems sind zuerst schrittweise komplexere Operationen (z.B. für die Datenbereitstellung <Wed79>) zu bilden, die dann die Grundlage zum Aufbau von Transaktionen sind. Diese komplexen Aktionsprozeduren (oder Datenbank-Aktionen) erfüllen zwar nicht die Transaktionseigenschaften, repräsentieren aber bereits komplexe anwendungsspezifische Operationen.

Entsprechend zu den Transaktionsprozeduren (aus Abs. 2.2.1) läßt sich für Aktionen der folgende Rahmen zur Spezifikation und Implementation angeben. Die Sprachkonstrukte, in der die Notation erfolgt, wurden an Modula-2 (<Wir82>,

<RRUZ82>) angelehnt. Es werden dabei einige Ergänzungen vorgenommen. In Abschnitt 3.2 erfolgt eine Zusammenfassung der einzelnen Aktionen mit den zugehörigen Relationen zu Moduln. Es ergibt sich dann die Möglichkeit gewisse Teile der Aktionsspezifikation modulspezifisch zu notieren.

```

ACTION SPECIFICATION    action-name
    ( cl: <type>; ...; ck: <type>;
      (* Liste von Eingabeparametern *)
      VAR ol: <type>; ...; ol: <type> );
      (* Liste von Ausgabeparametern *)
(* SCOPE *)
(* Angabe der benötigten Relationen *)
FROM DATABASE db-name IMPORT    <rell,...,reln>;
(* im Abs. 3.2 erfolgt die Erweiterung auf Sichten *)

(* DB-VARIABLES *)
(* Relationenschemata *)
    TYPE relationl : RELATION OF RECORD
                                attrl : <type>;
                                :
                                attrn : <type>

    END;
    :
(* Vereinbarung der Tupelvariablen *)
    RANGE OF rll ,..., rlm IS <relationl>;
    :
(* VARIABLES *)
(* weitere Typ- und Variablenvereinbarungen *)
TYPE .....
VAR ....

(* CONDITIONS *)
(* Angabe der Bedingungen *)

INVARIANT-CONDITION:    <predicate>;
(* Die Bedingungen sind vor und nach Ausführung der Aktion

```

gültig, die invarianten werden in Kapitel 5 den spezifischen Ebenen "Datenbank" und "Sicht" zugeordnet. *)

```
PRE-CONDITION:      <predicate>;
(* Vorbedingung, Voraussetzung zur Ausführung *)
POST-CONDITION:     <predicate>;
(* Nachbedingung, Angabe des Effekts nach der Ausführung *)

END ACTION action-name.
```

Die Implementation einer Aktion ist dann wie folgt aufgebaut:

```
ACTION IMPLEMENTATION  action-name (<parameter>);
(* Vereinbarungsteil analog zur Spezifikation *)
VAR ...
BEGIN-OF-ACTION;
(* Folge von Datenbankoperationen oder weiteren Aktionen
   ergänzt durch Anweisungen in z.B. Modula-2 *)
    :
    <db-operation>
    :
    <actions>
    :
    <db-operation>
    :
    <actions>
    :
END-OF-ACTION;
```

Der prinzipielle Aufbau einer Aktion ergibt sich aus der Trennung von Spezifikations- und Implementationsteil. Der Spezifikationsteil unterteilt sich wiederum in einen Vereinbarungsteil (hier als SCOPE bezeichnet) und den Bedingungen (CONDITIONS) zur Aktionsausführung.

Der Vereinbarungsteil stellt durch die IMPORT-Anweisung (analog zu Modula-2) die Verbindung zur Datenbank her. Es sind an dieser Stelle auch die für die Formulierung der Bedingungen notwendigen Typen und Variablen zu vereinbaren. Die invarianten Bedingungen beziehen sich auf die Datenbank (bzw. auf die Sicht) und sind sowohl als Teil der Vor- wie auch der Nachbedingungen zu sehen. Sie werden später (Kapitel 5) der Datenbank bzw. dem Externen Schema zugeordnet. (Auf die Anforderungen an die Spezifikation und Implementation, die durch Bildung Externer Schemata entstehen, wird in Kap. 3 näher eingegangen.)

Der Implementationsteil gibt einen Algorithmus an, der den durch Vor- und Nachbedingungen festgelegten Übergang realisiert. Für die Implementation können die üblichen Kontrollstrukturen und Anweisungen (wie z.B. Modula-2) ergänzt durch eine Einbettung einer Datenmanipulationssprache (z.B. QUEL) verwendet werden. Innerhalb der Implementation können auch weitere Aktionen aufgerufen werden.

Ein ausführliches Beispiel zum "Eintrag eines Arztbesuchs" findet sich in Anhang 2. Die Datenintegritätsbedingungen sind (vgl. Abs. 5.2) dabei schichtspezifisch der Datenbank und der Sicht für einen Außendienstmitarbeiter zu entnehmen.

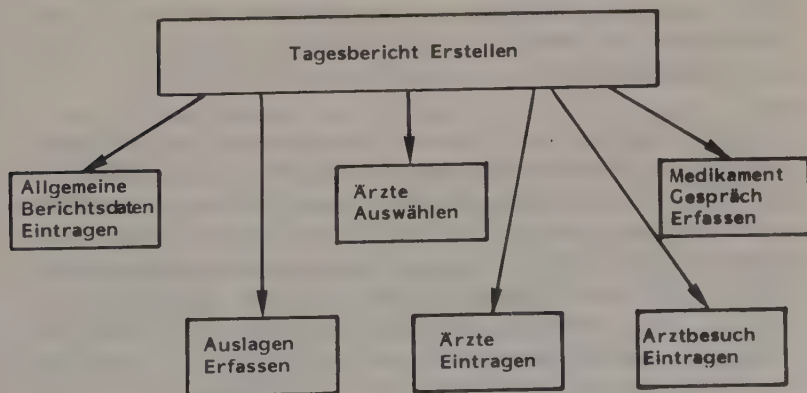
Die Spezifikation einer Aktion entspricht der im vorhergehenden Abschnitt angegebenen Spezifikation für Transaktionsprozeduren. Der wesentliche Unterschied besteht darin, daß die Ausführung einer Aktion nicht integritäts-erhaltend für die Datenbank ist. Die angegebenen Prädikate zur Invarianz des Datenbankzustands in den "Invariant-Conditions" enthalten folglich nicht alle durch die Aktion betroffenen Integritätsbedingungen.

Die invarianten Bedingungen der Datenbank, die nicht vor und nach der Aktion gelten, sind getrennt zu behandeln. Sie lassen sich letztlich nur bezüglich einer Transaktion spezifizieren und insbesondere überwachen (vgl. Abs. 5.5).

Diese vorgestellten (Datenbank-)Aktionen bilden im weiteren die Bausteine für den Aufbau von Transaktionen. Man erhält eine Konstitutionshierarchie (vgl. <Wed82>), die aus den einzelnen Aktionen verbunden durch die üblichen Kontrollstrukturen besteht. An der Spitze der Hierarchie kann letztlich eine Aktion stehen, die die Transaktionskriterien erfüllt - also eine Transaktion. Die Transaktionsprozedur ist somit ein Spezialfall der hier vorgestellten Aktionen. Wie zu erkennen ist, ist das in Abs. 2.2.1 skizzierte Beispiel "Eintrag eines Arztbesuch" in eigentlichen Sinne nur eine Aktionsprozedur. Sie wird von der Transaktion "Tagesbericht Erstellen" (vgl. dazu Anhang 2 und 3) benutzt.

Abb. 2-5:

(* vereinfachte Konstitutionshierarchie
am Beispiel "Tagesbericht Erstellen" *)



2.2.3. Die Transaktion als Prozeß

Die Menge der möglichen Datenbankzustände bilden einen Zustandsraum (vgl. Abs. 1.1.1). Eine Transaktion besteht aus einer Folge von Datenbank-Operationen. Sie überführt einen zulässigen Datenbankzustand über eine festgelegte endliche Menge von Zwischenzuständen in einen (neuen) Zustand. Neben dem Anfangs- und dem Endzustand einer Transaktion, werden auch diese Zwischenzustände der Datenbank (wenigstens teilweise) transparent, wenn man von reinen Batch-Transaktionen dialogorientierten Transaktionen übergeht. Anschaulich gesprochen bewegt sich die Transaktion in ihrem Ablauf von einem Zustand der Datenbank zum anderen. Jede dieser Zustandsänderungen unterliegt dabei gewissen Einschränkungen. Diese Einschränkungen werden durch die Operationsfolge der Transaktion beschrieben (vgl. Abs 1.2.2 und <Gra80>).

Die Transaktion bildet damit einen Prozeß (im Sinne der Systemprogrammierung, vgl. <Ker82>, S.38) auf der durch die Datenmanipulationsoperationen zur Verfügung gestellten virtuellen Maschine. (Die von Gray <Gra80> gegebene Transaktionsdefinition ist einfach in die von Keramidis (<Ker82>, S.38) gegebene Definition eines sequentiellen Prozesses zu überführen. Die "History" einer Transaktion nach Gray entspricht dabei einer initialen Berechnung nach Keramidis.)

Zwei Aspekte sind dabei wichtig. Erstens, die Bereitstellung von Datenbank-Aktionen erhöht das Abstaktionsniveau der dem Benutzer zur Verfügung gestellten virtuellen Maschine. Analog zu der in <Ker82>, S.71 beschriebenen Vorgehensweise kann durch das Bereitstellen von Moduln eine höhere operationale Sicht auf die Datenbank realisiert werden (vgl. dazu Abs. 3.2). Zweitens, ist bei einer Betrachtung der Transaktion als Prozeß nicht nur der Zustand der Datenbank sondern auch der Zustand der Miniwelt mit einzubeziehen. Eine Transaktion ist dann nur als der automatisierte Teil eines Handlungskomplexes zu verstehen.

Davies <Dav78> wies in der Darstellung des sog. "inline processing" darauf hin, daß damit auch in die Kontrollmaßnahmen die Miniwelt mit einzubeziehen ist. Die Gray'schen Kriterien haben letztlich für die Gesamthandlung auf der Datenbank und in der Miniwelt zu gelten.

Jedem Übergang auf der Datenbank von einem konsistenten Zustand in einen anderen ist somit ein Prozeß (oder Transaktion) zugeordnet, der auf einer abstrakten Maschine abläuft, die durch eine geeignete Menge von Datenbank-Aktionen beschrieben wird (vgl. Abs 3.2). Die Transaktion kann selbst wiederum durch einen oder mehrere Prozesse im Sinne des Betriebssystems realisiert werden (vgl. <Gra77>, S.421).

Kommunikation mit den Benutzern

Aus dem Zusammenhang zwischen Miniwelt und Transaktion (vgl. auch Abs. 2.1) ergeben sich Abhängigkeiten zwischen der Transaktion und dem Benutzer. Man kann die begleitende Handlung des Benutzers zu einer Transaktion als kooperierenden "Prozeß" interpretieren, der mit der Transaktion kommuniziert. Abb. 2-6 skizziert den schematischen Zusammenhang.

Die Operationen zur Kommunikation mit dem Benutzer werden in der Datenbank-Terminologie als DC-Operation (oder auch DC-Aktionen) (<Dav72>, <Gra77>) bezeichnet. Die DC-Aktionen sind in die Konstitutionshierarchie einer Transaktion ebenso einzufügen, wie die Datenbank-Aktionen. Die Zusammenfassung der DC-Aktionen zu Moduln zur Handhabung einer definierten Benutzerschnittstelle wird in Abs. 3.2 noch aufgegriffen (vgl. hierzu auch <HB80>).

Abb. 2-6.:

(* Kommunikation Benutzer - Transaktion *)

Benutzer		Transaktion
initiiert	----->	BEGIN-OF-TRANSACTION
	<-----	IN - REQUEST
Parameter Eingabe	----->	:
		<Database-Actions>
		:
Ausgabe	<-----	OUT - REQUEST
Entgegennahme	<-----	IN - REQUEST
Parameter Eingabe	----->	:
		<Database - Actions>
		:
Ausgabe	<-----	OUT - REQUEST
Entgegennahme		:
		END-OF-TRANSACTION

In der Regel wird eine Transaktion durch einen Benutzer initiiert, wenn ein gewisser Zustand in der Miniwelt vorliegt. Die Transaktion kann jedoch auch vom "System" initiiert werden, wenn die Voraussetzungen für die Ausführung ausschließlich im Zustand der Datenbank begründet sind. In diesem Fall kann die Transaktion die Kommunikation mit dem Benutzer aufnehmen.

Wechselwirkungen mit anderen Transaktionen

Neben der Kommunikation mit den Benutzern ist auch die Kommunikation zwischen Transaktionen denkbar. Diese wird aber wegen der geforderten gegenseitigen Isolation der Transaktionen durch die Gray'schen Kriterien ausgeschlossen. Die Spezifikation von Abhängigkeiten zwischen den Transaktionen hat ausschließlich über ihre Vorbedingungen zu erfolgen. Die Aktualisierung einer Transaktion kann abhängig gemacht werden von einem Zustand der Datenbank, der jeweils

durch eine "kooperierende" Transaktion zu erzeugen ist.

Die direkte Kommunikation unter Transaktionen erfordert nicht nur einen höheren Aufwand zur Kontrolle der Atomarität und den damit verbundenen Recovery-Maßnahmen (vgl. dazu <Gra77> und <BR81>). Sie stößt bei einer Einbeziehung des Zustands der Miniwelt (wie vorher skizziert) an prinzipielle Grenzen, da die Ausgabe von Information an einen Benutzer nur sehr bedingt rücksetzbar ist (vgl. auch <BR81>, S.123).

Über die Vorbedingungen einer Transaktion sollte es möglich sein, daß eine Transaktion einen Datenbankzustand erzeugt, der die Ausführung weiterer Transaktionen anstößt (vgl. Abs. 5.2).

Strukturierungsprinzip Atomarität

Eine Transaktion repräsentiert ganz oder teilweise eine Handlung der Miniwelt. Diese Handlung der Miniwelt unterliegt der Forderung der Atomarität (vgl. Abs. 2.1.4). Die Handlung wird als Transaktion realisiert auf der erwähnten abstrakten Maschine aus einer Folge von DB/DC-Aktionen. Die Folge der Aktionen einer Transaktion ist auf dieser Ebene als atomar zu kennzeichnen. Die Aktionen einer Transaktion werden deshalb eingeleitet durch die Operation BEGIN-OF-TRANSACTION (kurz BOT) und abgeschlossen durch END-OF-TRANSACTION (kurz EOT).

Die Klammerung von Aktionen durch BOT und EOT legt fest, welche Aktionen für die Recovery-Behandlung im Fehlerfall als eine Einheit zu betrachten sind. Die EOT Operation erzwingt zusätzlich über das Kriterium der Dauerhaftigkeit, daß die Wirkung der Transaktionen verbindlich und damit anderen Transaktionen zugänglich gemacht wird.

Zur Spezifikation und Implementation

Die Spezifikation der Transaktion erfolgt analog zu Abs. 2.2.1 in der Form von Vor- und Nachbedingungen. Die Schwierigkeit ist, daß der Voraussetzung zur Ausführung einer Transaktion sowohl vom Zustand der Datenbank als auch der Miniwelt abhängig ist. Der Zustand der Miniwelt kann aber erst während der Ausführung durch den Benutzer im Dialog übermittelt werden.

Die Spezifikation hat sich somit u.U. auf die Beschreibung des Zustandsübergangs der Datenbank zu beschränken. Die Möglichkeiten der Spezifikation von Transaktionen ist wesentlich von der Struktur des Übergangs abhängig. Im Kapitel 4 werden die verschiedenen Möglichkeiten erläutert. Beispiele für die prozeßorientierte Strukturierung von Transaktionen sind im Anhang 3 und 4 aufgeführt.

Zur Implementation von Transaktionen werden zwei Arten von Operationen benötigt:

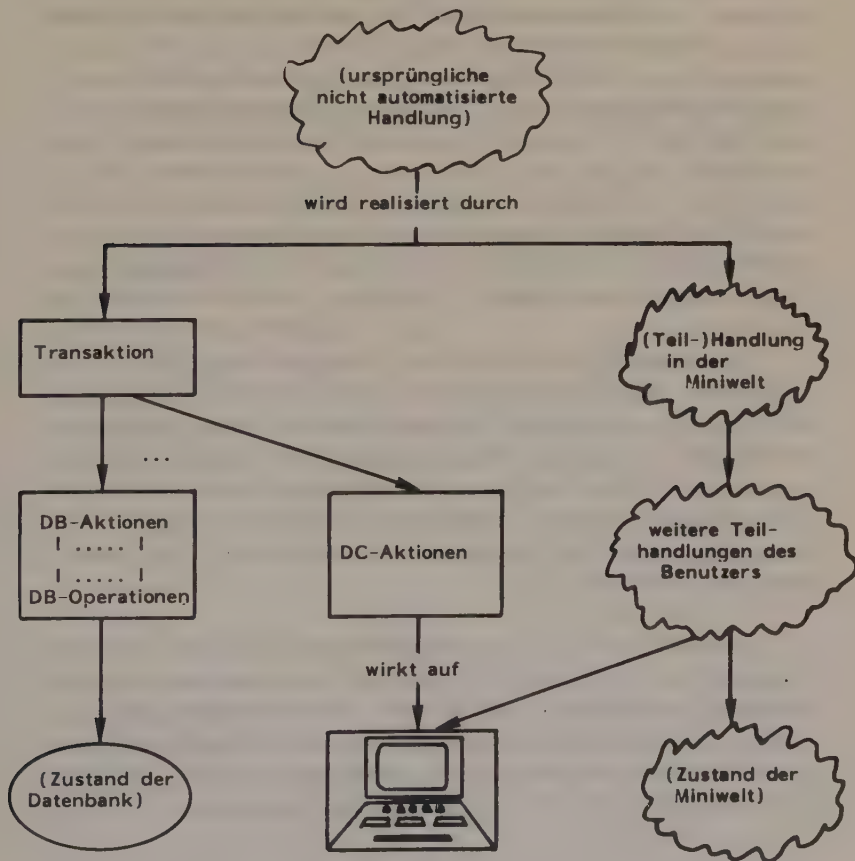
- (1) Datenbank-Operationen
(auch in der Form der vorgestellten Aktionen)
- (2) DC-Operationen zum Benutzer
(zur Kommunikation und Steuerung des benutzerbezogen Dialogs).

Die DC-Operationen dienen zum einen der Gewinnung der Parameterwerte als Zustandsbeschreibungen oder -änderungen der Miniwelt und zum anderen der Steuerung des Transaktionsablaufs (vgl. Kap. 4).

Eine Transaktion wird auf einer abstrakten Maschine mit DB/DC-Operationen als Bestandteil einer Gesamthandlung implementiert (vgl. Abb 2-7).

Abb. 2-7.:

(* Transaktionen als Bestandteil einer Gesamthandlung *)



3. Zum strukturellen Aufbau von Datenbank-Anwendungssystemen

Nach der isolierten Betrachtung der Struktur einzelner Aktionen und Transaktionen ist zu untersuchen, wie aus diesen ein Datenbank-Anwendungssystem als Ganzes zu strukturieren ist. Wie im einführenden Abschnitt 1.1.2 erwähnt wurde, wird einem Benutzer nicht die gesamte Datenbank, sondern nur eine für seine Belange relevante, spezifische Sicht zugänglich gemacht. Dieser begrenzte Ausschnitt der Datenbank kann als eine lokale, virtuelle Datenbank aufgefaßt werden. Die anwendungsspezifischen Operationen, realisiert als Aktionen, sind (zumindest virtuell) auf diesen Sichten der Datenbank auszuführen.

Die Sammlung von Operationen bezüglich einer Sicht der Datenbank ist in gewisser Weise als ein virtueller Abstrakter Datentyp aufzufassen. Die Realisierung dieser Abstrakten Datentypen ist jedoch datenbankspezifisch. Die Unterschiede zu herkömmlichen Abstrakten Datentypen hinsichtlich der Realisierung werden behandelt. Das Teilen gemeinsamer Objekte zwischen den Abstrakten Datentypen über die Datenbank ist zu beachten.

Daneben ist zur Aktivierung der Transaktionen und zur Übergabe der benötigten Parameter für den Benutzer eine geeignete Schnittstelle aufzubauen. Der wichtigste Gesichtspunkt ist dabei die Zusammenfassung von Transaktionen aufgrund ihres semantischen bzw. pragmatischen Bezugs zur Miniwelt und zu den zugehörigen Sichten der Datenbank.

Die folgenden vier Unterabschnitte befassen sich mit

- (1) dem Aufbau von Sichten der Datenbank,
- (2) den Operationen auf Sichten und deren Zusammenfassung zu Abstrakten Datentypen,
- (3) dem Aufbau eines Anwendungssystems unter der Einbeziehung des Transaktionskonzepts,
- (4) dem Vergleich von vorhandenen Ansätzen mit der vorgestellten Architektur eines Datenbank-Anwendungssystems.

Die Problemstellungen die sich aus der Gewährleistung der Integrität ergeben, werden erst im Kapitel 5 behandelt. In folgenden werden ausschließlich die notwendigen Grundstrukturen betrachtet.

3.1. Das Externe Schema als Basis für Aktionen

In diesem Abschnitt wird aufgezeigt, wie die Sichten einer Datenbank (mit den Möglichkeiten von Änderungsoperationen auf diesen Sichten) durch geeignete Mechanismen zu realisieren und welche Kriterien dabei zu beachten sind.

Definition von Sichten für Relationale Datenbanken

Die Beschreibung einer Sicht auf die Datenbank erfolgt analog zum Konzeptionellen Schema der Datenbank durch eine Menge von Relationenschemata (vgl. Abs. 1.1.2). Ein wesentlicher Unterschied liegt darin, daß die Sicht auf der Datenbank definiert wird. Die Wertebereiche der abgeleiteten Relationen beziehen sich auf die Datenbank. Die Sicht der Datenbank wird mit Hilfe einer Sichtbildungsfunktion unter Verwendung der Datenmanipulationssprache festgelegt. Die Sicht einer Datenbank ist selbst wieder eine lokale, virtuelle Datenbank.

Die folgenden Begriffsbestimmungen verdeutlichen den Zusammenhang zwischen Sicht und Datenbank.

(1) Abgeleitete Relation

Eine Relation $\underline{V}_i$ mit dem Relationenschema $V_i(A_1, \dots, A_n)$ wird als "abgeleitete Relation" bezeichnet, wenn eine Ableitungsfunktion $rep-i$ existiert, die auf einer Menge von Relationen einer Datenbank (Basisrelationen) definiert ist und zu jedem Zustand der Datenbank einen Wert der "abgeleiteten" Relation liefert. Die Struktur der abgeleiteten Relation $\underline{V}_i$ ist dabei durch das "abgeleitete" Relationenschema $V_i(A_1, \dots, A_n)$ festgelegt.

$rep-i : DATABASE \rightarrow VIEW-i$ mit

$rep-i(db) = \underline{V}_i$

Die Repräsentationsfunktionen können durch Anweisungen in der Datenabfragesprache angegeben werden.

(Beispiele zur Definition abgeleiteter Relationen sind im Anhang 2 enthalten).

(2) Sicht einer Datenbank

(auch als virtuelle Datenbank bezeichnet)

Die (lokale) Sicht VDB einer Datenbank DB besteht aus einer Menge abgeleiteter Relationen V_i , für die gilt:

Es existiert eine Sichtrepräsentationsfunktion der Art

$$\text{REP} : \text{DATABASE} \rightarrow \text{DB-VIEW}$$

mit $\text{REP}(\text{db}) = \text{vdb}$

(3) Sichtdefinition

Eine Sicht wird demnach festgelegt durch eine Familie von Abbildungsfunktionen zur Bildung abgeleiteter Relationen. Sie erzeugen aus der Datenbank eine virtuelle Datenbank. Eine Sicht auf die Datenbank wird beschrieben durch ein Externes Schema

$$\text{ES} = (V_1, \dots, V_m),$$

$$\text{REP} = \langle \text{rep}_i : \text{DATABASE} \rightarrow \text{VIEW}_i \rangle$$

wobei $(V_1, \dots, V_m)$ ein Tupel von Relationenschemata ist und $\text{REP} = \langle \text{rep}_i : \dots \rangle$ eine Familie zugehöriger Repräsentationsfunktionen zur Bildung der abgeleiteten Relationen der Sicht ist.

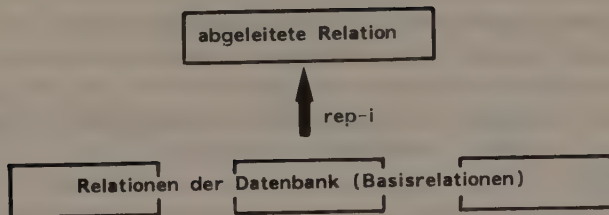
(4) Zustand einer Sicht

Der Zustand einer Sicht ist durch die Ableitungsfunktion oder Repräsentationsfunktion "REP" für jeden Zustand der Datenbank eindeutig bestimmt.

Hinweis: (In der Literatur wird häufig bereits eine abgeleitete Relation als Sicht bezeichnet, daher auch in QUEL das Sprachkonstrukt VIEW. In dieser Arbeit wird im allgemeinen unter einer Sicht eine Menge abgeleiteter Relationen im Sinne einer virtuellen Datenbank verstanden.)

Abb. 3-1:

(* Abgeleitete Relation *)

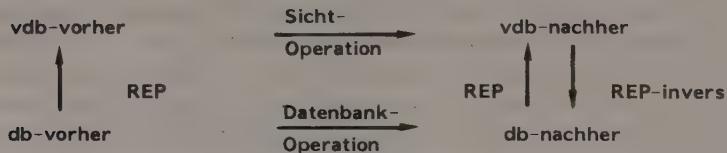


Das Transformationsproblem bei Änderungsoperationen

Würde man die Datenbank-Änderungsoperationen (in vorliegenden Fall QUEL - Anweisungen) einfach auf der Datenbanksicht anwenden, so ergibt sich die Schwierigkeit der eindeutigen Übersetzbarkeit, da die Repräsentationsfunktionen REP in der Regel nicht bijektiv sind.

Abb. 3-2.:

(* Repräsentationsfunktion und Inverse *)



Es besteht zwischen der Struktur der Datenbank (mit den relationalen Änderungsoperationen) und der virtuellen Datenbank (mit den relationalen Änderungsoperationen) nur ein Homomorphismus und kein Isomorphismus. Diesen Sachverhalt verdeutlicht Abb. 3-2 (vgl. dazu auch Paolini/Pegatti <PP77> und Keramidis <Ker82>, S.115). Es wurden mehrere Versuche unternommen, die Inverse (REP-inv) zur Abbildungsfunktionen aus Eigenschaften des Relationalen Datenmodells eindeutig zu bestimmen (vgl. <Zör82>). Vor allem über die Theorie der funktionalen Abhängigkeiten wurde versucht, die notwendige Zusatzinformation zur Festlegung der Eindeutigkeit zu erhalten <DB82>. Andere Ansätze <BS81>, <BS81a> berücksichtigen das Komplement der Sicht bezüglich der Datenbank, um auf eine eindeutige Inverse der Repräsentationsfunktion schließen zu können.

Insgesamt führte aber keiner der Ansätze zu einem befriedigenden, realisierbaren Ergebnis. Die Konsequenzen zog daraus Keller <Kel81>, indem er für bestimmte Klassen von abgeleiteten Relationen explizite Regeln zur Transformation der Änderungsoperationen angibt.

Die Umsetzbarkeit elementarer Änderungsoperationen von den Eigenschaften der virtuellen Datenbank auf die reale Datenbank ist abhängig von den Repräsentationsfunktionen, durch die die abgeleiteten Relationen definiert sind. Ergeben sich Mehrdeutigkeiten bezüglich der Inversen zur Repräsentationsfunktion, so ist die anwendungsspezifische Intension der Änderungsoperation zu berücksichtigen.

Es ist prinzipiell möglich, die Eindeutigkeit der Änderungsoperationen auf den abgeleiteten Relationen durch eine explizite Festlegung ihrer Übersetzung auf die Datenbank zu erzwingen (vgl. <RS79>). Der Ansatz von Keller <Kel81> ermöglicht, die Festlegung der Inversen durch das Bereitstellen geeigneter Alternativen zu unterstützen.

Die Art der Problemstellung für die Transformation der einzelnen Änderungsoperation (wie Einfügen, Ändern und Löschen von Tupeln, vgl. Abs. 1.2.1) ist abhängig von konkreten Eigenschaften der Repräsentationsfunktion. Aufgrund der Operationen der Relationenalgebra ergeben sich die im folgenden aufgezählten Arten der Sichtbildung. (Die Klassifikation berücksichtigt die Ergebnisse aus <Kel81>, <DB82>, <Zör82>).

Durch die Integritätsüberwachung ergeben sich u.U. weitere Einschränkungen. An dieser Stelle wird jedoch nur der Realisierungsaspekt betrachtet.

(1) Projektion

Zwei Arten der Projektion sind relevant:

(1.1) Identitätsprojektion

Die Schlüsselattribute der Basisrelation sind auch in der abgeleiteten Relation enthalten. Jedem Tupel der abgeleiteten Relation entspricht somit auch ein Tupel in der Basisrelation. Die zur abgeleiteten Relation gehörige Repräsentationsfunktion ist damit bijektiv. Dies ist der einfachste Fall zur Bildung abgeleiteter Relationen. Es ergeben sich die folgenden Transformationsmöglichkeiten.

- APPEND - Das Einfügen von Tupeln

Der Eintrag kann direkt erfolgen, da die Tupeln eindeutig in der Basisrelation bestimmt sind. Die in der abgeleiteten Relation nicht vorhandenen Attribute werden in der Basisrelation mit Nullwerten (undefinierten Werten) belegt.

(Zu Möglichkeiten der speziellen Behandlung undefinierter Werte (sog. Nullwerten) im Relationalen Datenbankmodell siehe z.B. <Cod79>, <Gol81>).

- DELETE - Löschen von Tupeln

Die zugeordneten Tupeln der Basisrelation werden gelöscht. Der Verlust von Attributwerten, die nicht zur Sicht gehören, kann nur durch ein Verbot des Löschens

verhindert werden. (Dies ist jedoch ein Aspekt der Datenbank-Integrität).

- REPLACE - Ändern von Tupeln

Die Änderungen erfolgen auf den in der abgeleiteten Relation übernommen Attributen der Basisrelation. Die anderen Attributwerte bleiben erhalten.

(1.2) Reduktionsprojektion

In die abgeleitete Relation werden die Schlüsselattribute der Basisrelationen nicht übernommen. Einem Tupel in der abgeleiteten Relation können somit mehrere Tupeln in der Basisrelation zugeordnet sein. Die Repräsentationsfunktion ist zwar total jedoch nicht injektiv.

- APPEND

Für ein einzutragendes Tupel, das in der abgeleiteten Relation nicht vorhanden ist, könnte man ein neues Tupel in die Basisrelation eintragen. Es würde allerdings dafür kein Schlüsseleintrag existieren. Eine spezifische Behandlung "undefinierter" Schlüsselwerte wäre erforderlich.

(Eine praktikable Möglichkeit ist in diesem Fall, eine konkrete Umsetzung der Operation anzugeben.)

- DELETE

Für die Löscheoperation gibt es folgende Alternativen. Erstens man löscht die qualifizierten Tupel auch in der Basisrelation. Inwieweit dies aus Gründen der Datenintegrität sinnvoll ist, ist anwendungsspezifisch zu entscheiden. Die zweite Möglichkeit besteht darin, die Werte der Attribute, die in die Repräsentationsfunktion eingehen, so zu modifizieren (z.B mit Nullwerten belegen), sodaß die qualifizierten Tupel nicht mehr in der Sicht erscheinen.

- REPLACE

Das Ändern von Tupeln ist eigentlich unproblematisch, es werden alle qualifizierten Tupel der Basisrelation modifiziert. Komplikationen können jedoch bei der Integritätskontrolle auftreten.

(2) Restriktion

Die Repräsentationsfunktion ist in diesem Fall injektiv und partiell. Die Problematik liegt in der Teilmengeneigenschaft der abgeleiteten Relation. Das Ansprechen eines Tupels im Komplement der abgeleiteten Relation erfordert die Angabe, ob es in der Basisrelation bereits enthalten ist.

- APPEND

Es ist zu prüfen, ob ein Tupeleintrag mit gleichem Schlüsselwert bereits in der Basisrelation vorhanden ist. Der Fall eines normalen Einfügens ergibt sich, wenn ein entsprechendes Tupel in der Basisrelation nicht existiert. Die Einfügeoperation ist auf der Basisrelation durch eine Änderungsoperation zu realisieren, falls auf ein Tupel mit gleichem Schlüsselwert bereits vorhanden ist.

- DELETE

Auch hier gibt es zwei Möglichkeiten. Entweder wird das Tupel gelöscht oder die Attributwerte, über die die Restriktion definiert ist, werden so modifiziert, daß das Tupel aus der abgeleiteten Relation verschwindet. Es verbleibt jedoch in der Basisrelation.

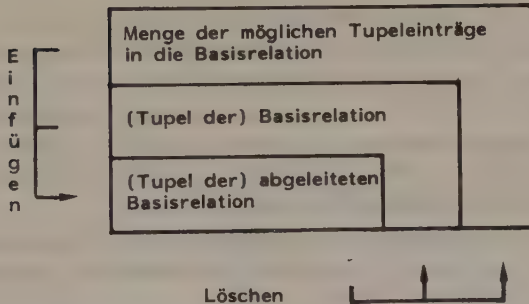
- REPLACE

Das Ändern ist in diesem Zusammenhang problemlos. Aus Gründen der Integritätswahrung ist u.U. mit den bekannten Mechanismen <Sto75> zu verhindern, daß durch Änderungen der Zustandsraum der abgeleiteten Relation verlassen wird.

Die Abbildung 3-3 verdeutlicht die Schwierigkeiten bei der Sichtbildung mittels Restriktion.

Abb. 3-3.:

(* Änderungen bei mittels Restriktion
abgeleiteten Relationen *)



(3) Verbund (beschränkt auf den Gleichheitsverbund)

Bei Repräsentationsfunktionen, die durch Verbundoperationen gebildet werden, stellen sich besondere Schwierigkeiten für Transformation von Änderungsoperationen ($\langle DB78 \rangle$, $\langle Kel81 \rangle$). Allgemeinere Möglichkeiten zur Umsetzung der Änderungsoperationen anzugeben, ist nur sehr eingeschränkt und mit sehr großem Aufwand möglich (vgl. $\langle DB78 \rangle$). Wie von Keller $\langle Kel81 \rangle$ gezeigt wurde, lassen sich für einige spezifische Arten des Gleichheitsverbunds (auch natürlicher Verbund) allgemeingültige Mechanismen angeben.

Es sind drei Arten des Gleichheitsverbunds zu unterscheiden:

(3.1) Identitätsverbund

Die abgeleiteten Relationen entstehen aus den Basisrelationen, indem Attribute mit Schlüsseleigenschaft verbunden werden. Jedem Tupel der abgeleiteten Relation ist somit eindeutig jeweils ein Tupel in den Basisrelationen zugeordnet. Die Repräsentationsfunktion ist bijektiv.

- APPEND, DELETE, REPLACE

Zur Übersetzung der Änderungsoperationen ist wegen der Bijektivität der Repräsentationsfunktion analog zur Identitätsprojektion vorzugehen.

(3.2) Direktverbund

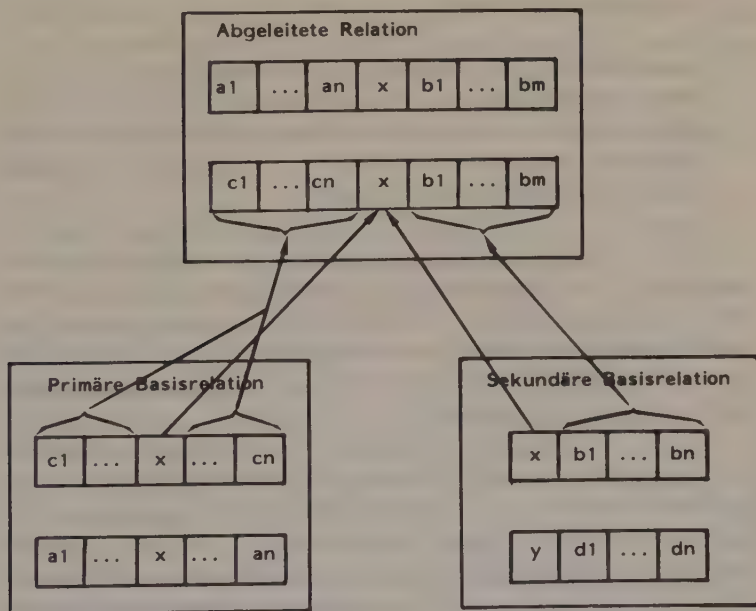
Der Schlüssel nur einer (primären) Basisrelation bleibt in der abgeleiteten Relation erhalten. Der Schlüssel der anderen (sekundären) Basisrelationen wird mit einem Nichtschlüsselattribut der primären Relation verbunden. Die durch erzeugte Relation ist nur in zweiter Normalform (vgl. <Cod71>). Die Abbildung 3-4 illustriert den Direktverbund zweier Basisrelationen.

Die Repräsentationsfunktion ist bezüglich der primären Basisrelation bijektiv, bezüglich der sekundären jedoch nicht injektiv und nur partiell.

Für die Änderungsoperationen ist zu unterscheiden, ob die modifizierten Attribute aus der primären Basisrelation oder aus der sekundären Basisrelation sind oder das Attribut ist, über das der Verbund gebildet wird. Die drei Arten von Attributen sind differenziert zu behandeln.

Abb. 3-4.:

(* Abgeleitete Relation mittels Direktverbund *)



Die Attribute (ausschließlich dem Verbundattribut) aus der primären Basisrelation können analog zur Identitätsprojektion behandelt werden. Die Attribute aus der sekundären Basisrelation erfordern eine explizite Angabe der Umsetzung. Ein Ändern des Verbundattributs bedingt u.U. eine implizite Veränderung weiterer sekundärer Attribute in der abgeleiteten Relation. Ob ein solches Vorgehen sinnvoll ist, ist anwendungsspezifisch zu entscheiden.

(3.3) allgemeiner Verbund

Es werden beliebige Nichtschlüsselattribute miteinander verbunden. Für diese Fälle abgeleiteter Relationen verbleibt nur eine explizite Angabe der Übersetzung der Änderungsoperationen.

(4) Kombinationen der einzelnen Mechanismen

Durch die Kombination der betrachteten Fälle wird die Behandlung der Änderungsoperationen auf den abgeleiteten Relationen komplexer. Es ergeben sich dabei allerdings keine wesentlichen neuen Aspekte.

Als Konsequenz zeigt sich, daß es in vielen Fällen notwendig ist, die Übersetzung der Operationen explizit anzugeben. Allerdings sind die skizzierten einfachen Fälle der Sichtbildung für viele Anwendungen hinreichend. Die Definition der Repräsentationsfunktion kann dann vom Datenbanksystem unterstützt werden.

Bei Sichtbildungsfunktionen, die sogenannte Verdichtungen der Daten durch in die Datenmanipulationssprache eingebettete Funktionen ("built-in-functions" <Cha76>, <Cha80>) zulassen, treten weitere Schwierigkeiten auf. In der Regel gilt für diese Fälle, daß eine explizite Angabe der Transformation der Operationen der Sicht erforderlich ist, wenn sie als sinnvoll erachtet wird.

Konstrukte zur Spezifikation des Externen Schemas

Die Spezifikation des Externen Schemas ist durch die Einführung geeigneter Sprachkonstrukte zu unterstützen. Die Angabe der Repräsentationsfunktion unter Berücksichtigung der ausgeführten Transformationsproblematik ist dabei zu berücksichtigen. Ein entsprechendes Vorgehen findet sich auch im Ansatz von Rowe/Shoens <RS79>. Die Beschreibung der Struktur eines Externen Schemas wird im folgenden an die Syntax von Modula-2 <Wir82> mit einer Einbettung von QUEL

<RT82a> angelehnt. Die notwendigen Erweiterungen, um den Datenbankbezug herzustellen, sind größtenteils selbst-erklärend bzw. werden im einzelnen im Anschluß noch erläutert. (Im weiteren der Arbeit wird allgemein die Syntax von Modula-2 <Wir82> mit einer Einbettung von QUEL <RT82a> zur Beschreibung der Aktionen, Transaktionen und Anwendungsmoduln verwendet.)

Für das Externe Schema ergibt sich folgende Struktur der Spezifikation:

```
EXTERNAL-SCHEMA virtual-database (<para1>; ...; <paraK>);
(* Definition einer virtuellen Datenbank *)
(* Die Parameter beeinflussen die Repräsentations-
   funktionen *)

SPECIFICATION;
(* Angabe der exportierten abgeleiteten Relationen *)
EXPORT PERSISTENT          view1, ..., viewM;
(* Zum Schlüsselwort PERSISTENT vergleiche die nachfolgende
   Erläuterung *)

(* Angabe der zum Aufbau der Sicht benötigten Relationen
   aus der Datenbank *)
FROM DATABASE database-name IMPORT PERSISTENT
    relation1, ..., relationN;

(* Festlegen der Struktur der Abgeleiteten Relationen *)
TYPE view1 = VIEW OF RECORD
    viewattr1l1 : <type>;
    :
    viewattr1lj : <type>;
    :
    viewattr1ln : <type>;
IDENT = (viewattr1l1,...)
END;
:
```

```

      :
TYPE viewM : VIEW OF RECORD
      viewattrM1 : <type>;
      :
      viewattrMj : <type>;
      :
      viewattrMn : <type>;
IDENT = (viewattrM1,...)
END;

REPRESENTATION;
(* Festlegen der Wertebereiche für die Tupelvariablen *)
(* für die abgeleiteten Relationen *)
RANGE OF t-var-view11 IS view1 VIEW;
      :
RANGE OF t-var-viewMQ IS viewM VIEW;

(* für die Relationen aus der Datenbank *)
RANGE OF t-var-re111 IS relation1 RELATION;
      :
RANGE OF t-var-re1NP IS relationN RELATION;

(* Festlegen der Abbildungsvorschrift *)
DEFINE VIEW view1      (target-list-view1)
      WHERE <view-qualification>;
      :
DEFINE VIEW viewM      (target-list-viewM)
      WHERE <view-qualification>;

```

OPERATION;

(* Festlegen der Übersetzung der Sicht-Operationen *)

```

DEFINE ON viewI VIEW
    REPLACE (target-list-viewI)
BY ON <relationI> RELATION
    REPLACE (target-list-viewI1)
        WHERE <qualification>
    :
    ON relationJ RELATION
    REPLACE (target-list-viewJ1)
        WHERE <qualification>
    :
    :
DEFINE ON viewM VIEW
    REPLACE (target-list-viewM)
BY ON relationS RELATION
    REPLACE (target-list-viewSM)
        WHERE <qualification>
    :
    ON relationT RELATION
    REPLACE (target-list-viewTM)
        WHERE <qualification>

SCHEMA-END virtual-database.

```

Die einzelnen Komponenten des Schemas zur Beschreibung eines Externen Schemas sind somit

- SPECIFICATION

In diesem Teil werden die abgeleiteten Relationen wie auch die zur Sichtbildung benötigten Basisrelationen aufgeführt.

- EXPORT-Klausel

In der EXPORT-Klausel werden die abgeleiteten Relationen angegeben, die den anwendungsspezifischen Operationen zugänglich gemacht werden.

Der Zusatz "PERSISTENT" bringt (vgl. <Atk78>) zum Aus-

druck, daß die Relationen der Datenbank permanent, d.h. über die Lebensdauer der sie importierenden Moduln erhalten bleiben. Daneben zeigt der Zusatz an, daß die Relationen der Datenbank von mehreren Sichten importiert werden können und eine entsprechende Synchronisation vorzunehmen ist. Die Relationen verhalten sich analog zu Modula-2 wie Variable, die von einem anderen Modul importiert werden.

- IMPORT-Klausel

In der IMPORT-Klausel erfolgt die Angabe der Basisrelationen der Datenbank.

- TYPE Vereinbarungen

Für die abgeleiteten Relationen ist ein Relationenschema anzugeben. Die geschieht in analoger Weise zum Konzeptionellen Schema der Datenbank, vgl. Anhang 1.

(Es wird nur anstelle der Bezeichnung "RELATION OF ..." die Bezeichnung "VIEW OF ..." verwendet. Wie im Beispiel demonstriert können dann die gleichen Namen sowohl für die abgeleitete Relation als auch die Basisrelation verwendet werden.)

- REPRESENTATION

In diesem Teil werden die Repräsentationsfunktionen für die einzelnen abgeleiteten Relationen angegeben.

- RANGE Angaben

Durch die RANGE Angaben werden die zur Formulierung der Repräsentationsfunktionen und Operationen notwendigen Tupelvariablen vereinbart.

- DEFINE VIEW ...

Durch diese Vereinbarung erfolgt die eigentliche Angabe der Repräsentationsfunktion für eine abgeleitete Relation durch das entsprechende Sprachkonstrukt aus QUEL.

- OPERATION

Der Abschnitt OPERATION legt die Transformation der sichtspezifischen Operationen auf allgemeine Datenbank-Operationen fest.

Die Spezifikation zur Sichtbildung wird durch das folgende Beispiel aus der Pharma-Datenbank veranschaulicht. Es wird die Sicht bezüglich eines Außendienstmitarbeiters gebildet. (Das Beispiel findet sich in ausführlicher Form im Anhang 2).

EXTERNAL-SCHEMA AERZTEBESTAND (Mitarbeiter);

(* Definition einer virtuellen Datenbank zur Verwaltung
des Ärztebestands eines Außendienstmitarbeiter *)

SPECIFICATION;

(* Angabe der exportierten abgeleiteten Relationen *)

EXPORT PERSISTENT ARZT;

(* Angabe der zum Aufbau der Sicht benötigten Relationen
aus der Datenbank *)

FROM DATABASE PHARMADATENBANK IMPORT PERSISTENT

ARZT, ARZTBESUCH, MITARBEITER, PHARMAGEBIET;

(* Festlegen der Struktur der Abgeleiteten Relationen *)

TYPE ARZT = VIEW OF RECORD

ArztNr : CARDINAL;

:

RPM-KreisNr : CARDINAL;

:

Anzahl-Besuche : CARDINAL;

IDENT = (ArztNr)

END;

(* Die abgeleitete Relation ARZT wurde um einige Attribute
aus der Relation ARZTBESUCH erweitert *)

REPRESENTATION;

(* Festlegen der Wertebereiche für die Tupelvariablen *)

(* für die abgeleiteten Relationen *)

RANGE OF arztview IS ARZT VIEW;

(* für die Relationen aus der Datenbank *)

RANGE OF arzt IS ARZT RELATION;

:

RANGE OF arztbes2 IS ARZTBESUCH RELATION;

(* Restriktion der Ärzte auf das Mitarbeitergebiet *)

DEFINE VIEW ARZT

(ArztNr=arzt.ArztNr, ...,

WHERE mitarbeit.MA-Nr = Mitarbeiter

AND gebiet.OrgEinheit = mitarbeit.MA-KreisNr

AND gebiet.PhGeNr = arzt.RPM-KreisNr

AND

OPERATION;

(* Festlegen der Übersetzung der Sicht-Operationen *)

DEFINE ON ARZT VIEW APPEND, DELETE

BY STANDARD ON ARZT RELATION;

(* Die aus ARZTBESUCH abgeleiteten Attribute werden
ignoriert *)

DEFINE ON ARZT VIEW

REPLACE (Anrede=arztview.Anrede, ...,

Besuchsbemerkung=arztview.Besuchsbemerkung)

BY ON ARZT RELATION

REPLACE (Anrede=arztview.Anrede, ...,

WHERE arzt.ArztNr = arztview.ArztNr

ON ARZTBESUCH RELATION

REPLACE (Bemerkung=arztview.Besuchsbemerkung)

WHERE arztbes2.ArztNr=arztview.ArztNr

AND

SCHEMA-END AERZTEBESTAND.

Aus dem Beispiel ist auch zu ersehen, daß die "Parametrisierung" von Sichten sinnvoll ist. Die Sicht wird spezifisch für jeden Außendienstmitarbeiter auf identische Art und Weise gebildet. Durch die Einführung parametrisierter Sichten vereinfacht sich ihre Verwaltung wesentlich. Die Eingabe der jeweiligen Personalnummer des Mitarbeiters ist allerdings von der Sicherheitskomponente des Datenbank-Anwendungssystems zu überwachen. (Zur Verwaltung parameterisierter Sichten siehe auch <MS83>).

3.2. Aufbau einer operationalen Sicht der Datenbank

Nach den Betrachtungen der statischen Komponente eines Datenbank-Anwendungssystems kann der Aufbau der operationalen Komponenten erfolgen. Nach der Bildung von Sichten ist die Frage zu beantworten, wie die einzelnen Aktionen zu Einheiten zusammenzufassen sind. Das Ziel ist es, das Konzept des Abstrakten Datentyps (vgl. Jones <Jon78>) als Strukturierungsmittel einzusetzen. In der Literatur finden sich bereits einige Ansätze zur Verwendung Abstrakter Datentypen zum Aufbau von Datenbanken. Es sind an dieser Stelle die Ansätze von Weber, Paolini u.a. (<Web78>, <EKW78>, <PP77>, <Pao78>) zu nennen. Sie versuchen jedoch dem Datenbanksystem eine völlig andere Struktur zugrunde zu legen und gelangten deshalb auch zu keiner praktischen Relevanz. Ein neuerer, interessanter Ansatz stammt von Rebsamen/Zehnder et. al <RZ82>, <RRUZ82>, der im Abs. 3.4 mit dem im folgenden gemachten Vorschlag verglichen wird.

Virtueller Abstrakter Datentyp

Auf den durch die Externen Schemata definierten virtuellen Datenbanken lassen sich die Aktionen realisieren, wie sie Abs. 2.2.2 eingeführt wurden. Die Zusammenfassung von Aktionen bezüglich einer virtuellen Datenbank und den spezifischen Relationen bildet einen Abstrakten Datentyp. Einem solchen Abstrakten Datentyps sind seine Objekte allerdings nur virtuell zugeordnet, da die gleichen Objekte einer Datenbank mehreren dieser ADT's zugeordnet sein können. Der aktuelle Zustand der Objekte eines ADT's kann somit nicht aus der Vorgeschichte allein "seiner" Operationen bestimmt werden. Diese Art Abstrakter Datentypen soll deshalb als Virtueller Abstrakter Datentyp (VADT) bezeichnet werden. Im Abs 5.4 wird der Virtuelle Abstrakte Datentyp noch um Komponenten zur Kontrolle der Integrität erweitert.

Der hier eingeführte Aufbau ist zwar in der Schnittstelle zum Benutzer dem Vorschlag von Weber <Web78> ähnlich. Er unterscheidet sich aber prinzipiell, da Weber als Basiselemente für den Aufbau der Datenbank Moduln verwendet, während der hier gemachte Vorschlag, auf der Erweiterung der Sichten basiert.

Ein Virtuellen Abstrakten Datentyp enthält eine Menge abgeleiteter Relationen (die durch ein Externes Schema festgelegt sind) mit den zugehörigen Aktionen (anwendungsbezogenen Operationen). Die Beziehungen zwischen den Relationen und die korrekte Ausführung der Operationen wird durch die Integrität gewährleistet (vgl. Abs. 5.4).

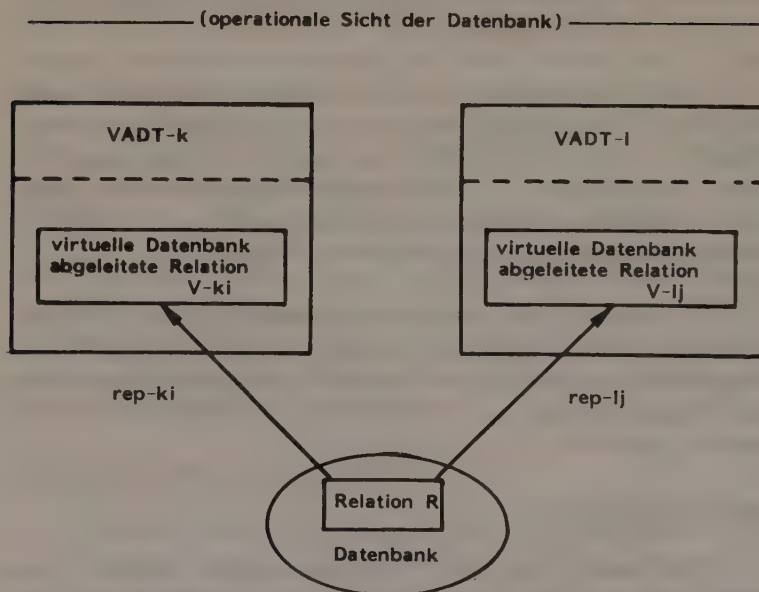
Im weiteren wird dieses durch anwendungsspezifische Aktionen erweiterte Externe Schema auch als Datenbank-Anwendungsmodul bezeichnet. Die Transaktionen werden entsprechend der Vorgehensweise aus Abs. 2.2.3 als Prozesse auf diesen Moduln formuliert.

operationale Sicht

Die Operationen der Anwendungsmoduln bilden eine "operationale" Sicht der Datenbank. Dieser Sachverhalt wird in Abb. 3-5 verdeutlicht. Diese operationale Sicht ist die abstrakte Maschine, auf der die Transaktionen eines Anwendungssystems zu realisieren sind. Für den Aufbau der VADT's wurde folgende Vereinfachung vorgenommen: Jeder Sicht der Datenbank wurde genau ein Virtueller Abstrakter Datentyp zugeordnet, durch den die Änderungsoperationen auf der Sicht ausgeführt werden. Die Verwendung anderer Sichten in einem Virtuellen Abstrakten Datentyp erfolgt nur lesend und ist den Benutzern des VADT nicht transparent. Auf weitere Sichten wird nur zurückgegriffen, soweit sie für die Realisierung der Aktionen des VADTs erforderlich sind.

Abb. 3-5.:

(* Operationale Sicht einer Datenbank *)



Die operationale Sicht einer Datenbank wird demnach durch eine Menge von Anwendungsmoduln gebildet. Im hier gemachten Vorschlag bleibt (im Gegensatz zu <Web78>) die Verwaltung der Datenobjekte mittels eines Relationalen Datenbanksystems erhalten. Die lokale Sicht für einen Benutzer (oder für eine Transaktion) ergibt sich somit aus der Menge, der ihm zur Verfügung stehenden Anwendungsmoduln.

Abhängigkeit zwischen den VADT's

Aus der Abb. 3-5 ist zu entnehmen, daß die Menge der Objekte der einzelnen Anwendungsmoduln nicht disjunkt ist. Durch die Tatsache, daß sich der Zustand der Objekte eines Virtuellen Abstrakten Datentyps nicht ausschließlich aus der Geschichte der ausgeführten Operationen ableiten läßt, weisen die Anwendungsmoduln ein nicht deterministisches Verhalten auf.

Auf diesem Sachverhalt beruht ein wesentlicher Unterschied zu nicht datenbankgestützten Anwendungssystemen. Die Kommunikation zwischen den einzelnen Moduln erfolgt nicht operational durch das Austauschen von Nachrichten, sondern statisch durch das Teilen gemeinsamer Objekte in der Datenbank. Es ergibt sich somit ein impliziter Informationsfluß, der durch die Repräsentationsfunktionen bei der Sichtbildung bestimmt wird.

Spezifikation von Anwendungsmoduln

Die Datenbank-Anwendungsmoduln sind in ihrer Struktur zu definieren. Die Notation zur Spezifikation wird dabei an Modula-2 und QUEL angelehnt. Wie bereits erwähnt sind hierfür zusätzliche Konstrukte zur Herstellung des Datenbankbezugs und zur Formulierung der Vor- und Nachbedingungen einzuführen.


```
DB-APPLICATION-MODULE appl-module (<parameter-list>);
SPECIFICATION;

EXPORT QUALIFIED          <action-list>;
                           <rec-typ-list>;

IMPORT PERSISTENT virtual-database;
(* Sicht der Datenbank, wie in Abs. 3.1 beschrieben *)
:

ACTION aktion-name (<parameter-list>);
(* Festlegen der Aktionen *)
(* analog zur Beschreibung in Abs. 2.2.2 *)

PRE-CONDITION: <predicate>;
(* Angabe der Vorbedingung zur Ausführung der Aktion *)
POST-CONDITION: <predicate>;
(* Angabe des Effekts der Aktion auf der Datenbank *)

ACTION ....

:
MODULE-END appl-module.
```

Die Prädikate zur Spezifikation der Vor- und Nachbedingungen sind für die Datenbank mit Hilfe der Datenmanipulationssprache (hier QUEL) angebbbar, indem man Zustände von Relationen (gegebenfalls unter Anwendung eingebetteter Funktionen) vergleicht. In den Nachbedingungen kann die Wirkung der Operation auf die Sicht durch einen Vergleich des Zustands der Relationen vor und nach der Ausführung der Aktion spezifiziert werden. Durch das Anfügen des Suffix "-OLD" bzw. "-NEW" am Relationennamen bei der Vereinbarung der Tupelvariablen wird angegeben auf welchen Zustand sich die Variable in der Bedingung bezieht.

Als einfaches Beispiel ergibt sich der Eintrag eines Arztes in die abgeleitete Relation ARZT folgt:

```

DB-APPLICATION-MODULE ArztVerwaltung
                                (Mitarbeiter: CARDINAL);

SPECIFICATION;

EXPORT QUALIFIED                AerzteEintragen, AerzteAustragen,
                                AerzteAendern, ... Arzt;
IMPORT PERSISTENT AERZTEBESTAND (Mitarbeiter);
(* Sicht der Datenbank, wie in Teil 1 beschrieben *)

TYPE Arzt = RECORD   ArztNr      : CARDINAL;
                    :
                    Anzahl-Besuche : CARDINAL;

END;
(* zu den anderen Attributen vergleiche die Definition
   der abgeleiteten Relation Arzt in Anhang 2 Teil 1 *)
TYPE Aerzte = SET IN UNIVERSE OF Arzt;
(* das SET-Konstrukt wird entsprechend der von Grosch
   <Gro82> vorgenommen Erweiterung von Modula-2
   verwendet *)
:
VAR arztvar      : Arzt;
VAR arztmenge    : Aerzte;
:
RANGE OF arzt IS ARZT-NEW;
:
ACTION AerzteEintragen (arztmenge: Aerzte;
                        VAR aerztenr: NummernMenge);
(* es können mehrere Teile des Records in den Elementen der
   Variablen "arztmenge" unbesetzt sein. *)
VAR NeueArztNr : SET IN Nummernmenge;
    neuearztNr  : CARDINAL;
(* Die Arztnummer wird von der Aktion vergeben *)

```

PRE-CONDITION:

(* CONDITION *)

(FORALL arztvar IN arztmenge : arztvar.ArztNr = NIL)

AND

(* CONDITION ON ARZT *)

(FORALL neuearztNr IN NeueArztNr :

(NOT ANY (arztold.ALL)

WHERE neuearztNr = arztold.ArztNr));

POST-CONDITION:

(* CONDITION *)

(aerztenr = NeueArztNr)

AND

(FORALL arztvar IN arztmenge :

EXISTS1 neuearztNr IN NeueArztNr :

arztvar.ArztNr = neuearztNr)

AND

(* CONDITION ON ARZT *)

(FORALL neuearztNr IN NeueArztNr :

(COUNT (arzt.ALL)

WHERE neuearztNr = arzt.ArztNr

AND arztvar.Name = arzt.Name

:

AND arztvar.RPM-Kreis = arzt.RPM-Kreis)

= 1));

AND

(arzt.ALL) WHERE neuearztNr != arzt.ArztNr

=

(arztold.ALL)

(* != steht für "ungleich". *)

ACTION-END;

ACTION AerzteAustragen (arztmenge: Aerzte);

:

:

MODULE-END ArztVerwaltung.

Mengenorientierte operationale Schnittstelle

Im angegebenen Beispiel wie auch im Anhang 2, wurden die Änderungsoperationen der operationalen Sicht mengenorientiert angegeben. Die Angabe tupelorientierter Änderungsoperationen ist grundsätzlich ebenfalls möglich. Da die Möglichkeiten der gewählten Wirtssprache Modula-2 zur Mengenbearbeitung sehr eingeschränkt sind (vgl. <Wir82>, S.65), wurde zur Formulierung der mengenorientierten Schnittstelle auf den Erweiterungsvorschlag von Modula-2 durch Grosch (<Gro82>, S. 82ff) zurückgegriffen. Diese Vorgehensweise bietet eine adäquate Möglichkeit zur Erläuterung der operationalen Sicht einer Datenbank. *)

Für die Darstellung der Leseoperationen, die auf der operationalen Sicht zur Verfügung zu stellen sind, stellt sich die Frage nach der Mengenorientierung in analoger Weise. Für das Lesen einer Tupelmenge ist es gegebenenfalls wünschenswert, eine mengenorientierte Auswahl von Tupeln vorzunehmen und die einzelnen Tupel dieser Menge dann nacheinander weiter verarbeiten zu können. Hierfür bietet sich zum einen die Möglichkeit des sogenannten "Cursors" (vgl. <Cha76>, <Cha80>), den man über eine qualifizierte Menge von Tupeln legen kann. Die einzelnen Elemente der Menge können dann sequentiell über ihn gelesen werden. Zum anderen ist es möglich, analog zu EQUOL von INGRES (vgl. <RT82a>), eine Laufschleife zur Abarbeitung der Tupelmenge an die Auswahloperation zu binden.

*) Die Bearbeitung von Mengen von Tupel könnte man in Modula-2 neben der Spracherweiterung, wie sie von Grosch <Gro82> vorgenommen wurde, auch durch Einführen neuer Moduln zur Mengenbearbeitung auf der Basis von "Arrays" oder durch eine Erweiterung des vorhandenen SET-Konstrukts erreichen. Beides berührt jedoch die betrachtete Problematik der VADT's nur am Rande.

Im betrachteten Zusammenhang ist zu entscheiden, ob die Verwaltung des "Cursors" in den Datenbank-Anwendungsmoduln erfolgen soll oder ob eine Übergabe der gesamten, durch eine Abfrage qualifizierten Tupelmenge, wie im Beispiel erfolgt, an den Benutzer oder Transaktion vorzuziehen ist. (Das Cursor-Konzept entspricht der "FORALL <element> IN <menge> DO <anweisungen> END;" Lauschleife in <Gro82>, S.92.)

Beispiel:

Suche nach allen Ärzten, die ein bestimmtes Kriterium (z.B. Internist) erfüllen:

(1) Mengenverwaltung in der operationalen Sicht

durch die Bereitstellung der Operationen:

ArztAuswahl (Bedingung : PREDICATE);

(z.B. Arzt.Fach = Internist)

und

ArztElement (VAR arztvar : Arzt);

(2) Übergabe von Mengen

durch die Operation:

ArztAuswahl (Bedingung : PREDICATE,

VAR arztmenge : Aerzte);

oder ersatzweise

(beim Fehlen der mengenorientierten Erweiterung)

ArztAuswahl (Bedingung : PREDICATE,

VAR arztmenge : ARRAY [1..N] OF Arzt);

Welche Alternative verwendet wird, ist anwendungsspezifisch zu entscheiden. Die Entscheidung ist nicht zuletzt von den Möglichkeiten der Wirts- und der Datenmanipulationssprache abhängig zu machen.

Zur Implementation von Anwendungsmoduln

Entsprechend der Vorgehensweise in Modula-2 ist für den spezifizierten Anwendungsmodul ein Implementationsmodul anzugeben, der die spezifizierten Aktionen durch Operationen des zugehörigen Externen Schemas realisiert.

```
DB-APPLICATION-MODULE appl-module (<parameter-list>);  
IMPLEMENTATION;
```

```
EXPORT QUALIFIED  <action-list>,  
                  <rec-typ-list>;  
IMPORT PERSISTENT virtual-database;
```

```
ACTION action-name (<parameter-list>);  
(* Implementation der Operationen *)  
(* analog zur Beschreibung in Abs. 2.2.2 *)
```

```
:  
MODUL-END appl-module.
```

Ausgeführte Beispiele für den Aufbau von Anwendungsmoduln finden sich im Anhang 2.

Integration des Externen Schemas in den Anwendungsmodul

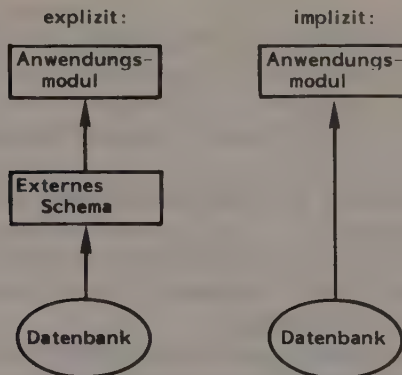
Für den Aufbau von Datenbank-Anwendungssystemen ergibt sich als Konsequenz:

- (1) Eine Trennung der Datenrepräsentationsfunktion von der Datenbank zur Sicht als erster Konkretionsschritt hin zur spezifischen Anwendung;
- (2) Der Aufbau einer operationalen Ebene auf diesen lokalen virtuellen Datenbanken durch das Konzept der Virtuellen Abstrakten Datentypen als zweiter Konkretionsschritt.

Wie aus Abs. 3.1 zu erkennen ist, ist eine wesentliche Schwierigkeit bei der Bildung von Sichten, die Eindeutigkeit der Inversen zur Repräsentationsfunktion zu erzwingen. Neben der im Abs. 3.1 aufgezeigten Möglichkeit gibt es die Alternative, die Operationen auf der Sicht nur vorzudefinieren, jedoch die Aktionen der Datenbank-Anwendungsmoduln direkt durch Operationen auf die Datenbank zu realisieren. Dieses Vorgehen bedeutet zwar einen Verlust der Datenunabhängigkeit, die durch die explizite Definition der Externen Schemata erreicht wurde, vereinfacht allerdings die Implementierung der Aktionen wesentlich. Auf diese Weise wird es generell möglich, die in Abs. 3.1 behandelte Problematik der Transformation von Operationen von den Sichten auf die Datenbank zu umgehen, wenn den Benutzern ausschließlich Aktionen auf dem Niveau der Datenbank-Anwendungsmoduln zur Verfügung gestellt werden.

Abb. 3-6.:

(* Explizite und implizite Angabe Externer Schemata *)

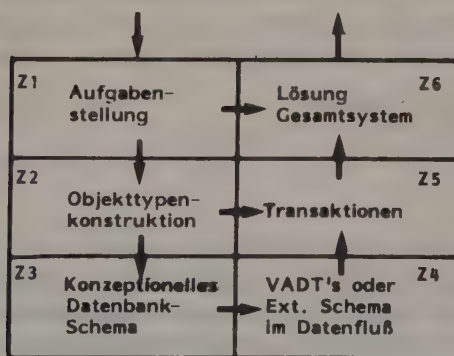


Einbettung in die Methodologie zum Datenbankentwurf

Der dargelegten Vorgehensweise liegt eine konsequente Anwendung der Wedekind'schen Gesamtmethodologie zur Erstellung von Datenbank-Anwendungssystemen zugrunde. Es werden die Zustandsübergänge Z3 (Konzeptionelles Schema der Datenbank) über Z4 (Externes Schema im Datenfluß) zu Z5 (Programmmentwurf) behandelt (vgl. <Wed79a>). Durch die dargelegte Vorgehensweise wird die Voraussetzung für eine datenbank-adäquate Transaktionsprogrammierung erreicht. Die Trennung zwischen Datenbereitstellungsteil (hier realisiert als Virtueller Abstrakter Datentyp) und Verarbeitungsteil in den Datenbank-Anwendungsprogrammen <Wed79> ergibt sich als natürliche Konsequenz.

Abb. 3-7.:

(* Einbettung der Vorgehensweise in die Gesamtmethodologie *)



3.3. Transaktionsorientierte Strukturierung eines Datenbank-Anwendungssystems

Die Transaktionen werden auf den Datenbank-Anwendungsmoduln mit Hilfe der zur Verfügung gestellten Aktionen gebildet. Die Trennung zwischen Bereitstellungs- und Verarbeitungsteil, wie sie von Wedekind <Wed79> gefordert wurde, wird hierbei vollzogen. Die Datenbereitstellung erfolgt in den Datenbank-Anwendungsmoduln, die Verarbeitung in den Transaktionen.

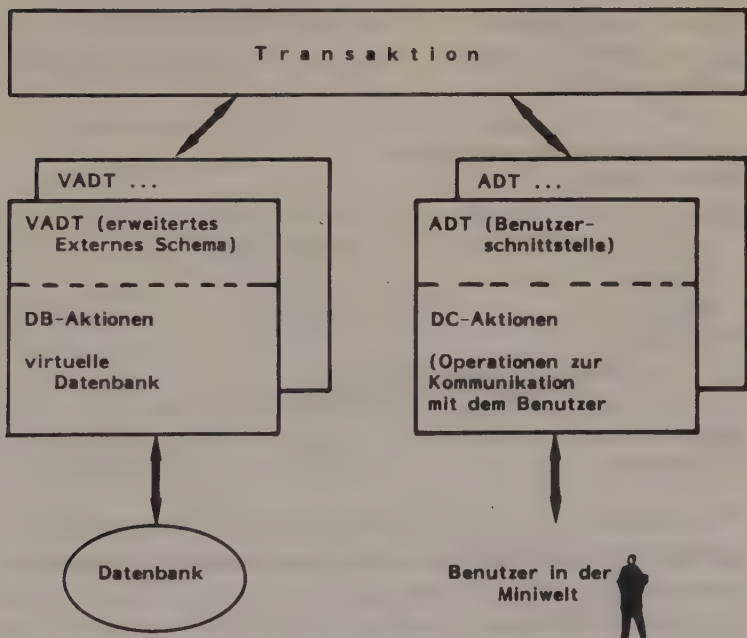
Eine Transaktion arbeitet somit auf einer operationalen Sicht der Datenbank. Die operationale Sicht kann aus mehreren Virtuellen Abstrakten Datentypen bestehen. Neben der operationalen Sicht auf die Datenbank sind für die Kommunikation mit dem Benutzer entsprechende Moduln bereitzustellen (hier als DC-Moduln bezeichnet). Sie übernehmen die Kontrolle des Datentransfers vom und zum Benutzer *). Einen Ansatz für die Zusammenfassung von DC-Aktionen zu DC-Moduln könnte, z.B. von den von Hammer/Berkowitz (<HB80>, S.81) eingeführten "ports" ausgehen **). Eine Transaktion arbeitet damit auf zwei Arten von Moduln, Datenbank- und DC-Moduln.

*) Der spezielle Aufbau von DC-Moduln wird nicht behandelt. Untersuchungen zum Aufbau von Dialogsystemen, im Rahmen derer die Strukturierung von DC-Moduln aus DC-Aktionen zu behandeln ist, sind nicht Gegenstand dieser Arbeit.

**) Hammer/Berkowitz führen zur Verwaltung des Benutzerdialogs in einer Datenbankmanipulationssprache zum semantischen Datenmodell "SDM" sogenannte "ports" ein. In den "ports" wird der Aufbau der für den Benutzer aufbereiteten und der von ihm einzugebenden Daten beschrieben.

Abb. 3-8.:

(* Transaktion auf DB-/DC-Moduln *)



Die Notation der Transaktionen erfolgt wiederum in der üblichen Vorgehensweise, durch eine Trennung von Spezifikation und Implementation. Es kann die in Abs. 2.2.3 gewählte Form beibehalten werden. Sie wird im Kapitel 4 ergänzt, wobei dort die Schematisierbarkeit von Transaktionen, und damit die Spezifizierbarkeit beim Erstellen eines Datenbank-Anwendungssystem im Mittelpunkt steht.

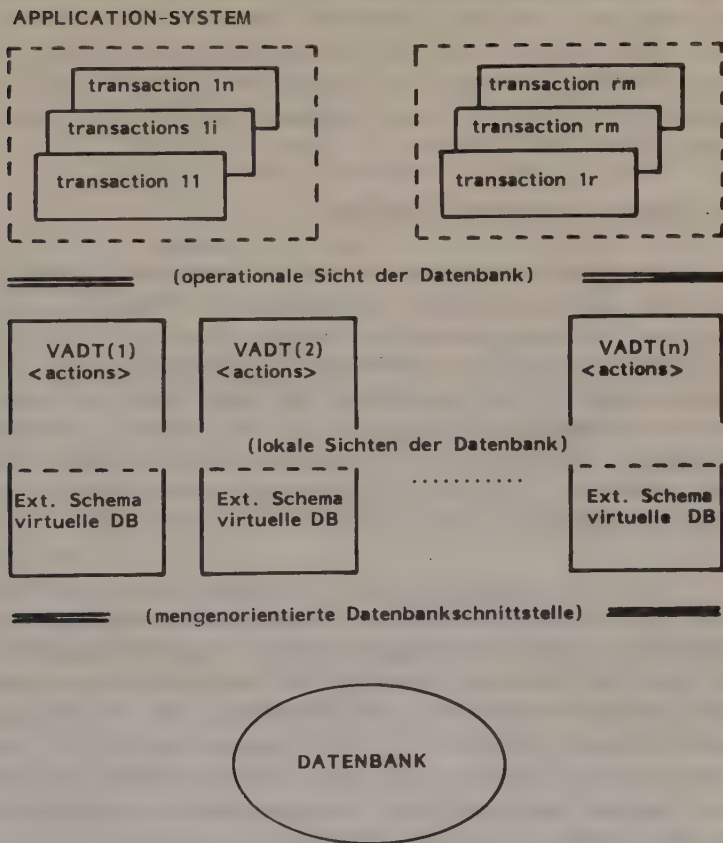
Die Transaktionen, die die für einen Benutzer möglichen Anwendungsoperationen umfassen, werden auf mehreren Anwendungsmoduln realisiert und lassen sich zu einem lokalen Anwendungssystem zusammenfassen. Diese Menge von Transaktionen bildet die für einen Datenbankbenutzer relevante Umgebung. Die benutzer- bzw. anwendungsspezifische Zusammenfassung von Transaktionen ergibt ein lokales Datenbank-Anwendungssystem.

```
APPLICATION-SYSTEM appl-system;
SPECIFICATION;
SCOPE;
    IMPORT PERSISTENT <application-modul-list>;
    TYPE ...
    VAR ...
TRANSACTIONS;
    <transaction specification>
    (* Spezifikation der einzelnen Transaktionen *)
SYSTEM-END appl-system;
```

Die Menge aller lokalen Anwendungssysteme bilden letztlich das (allgemeine) Datenbank-Anwendungssystem (oder auch Transaktionssystem). Der prinzipielle Aufbau eines Datenbank-Anwendungssystems aus der Sicht der Datenbank (ohne Kommunikation nach außen) wird durch Abb. 3-9 illustriert.

Um zu einer integrierten Verwaltung der einzelnen Schichten eines Datenbank-Anwendungssystems zu gelangen, sind noch vielfältige Anstrengungen bezüglich der Erweiterung des Data Dictionary zu unternehmen. In das erweiterte Data Dictionary sind neben den Datenbeschreibungen und den Integritätsbedingungen auch die Beschreibungen der Transaktionen (Export-Import-Listen, Funktionsaufrufe und Vor-, Nachbedingungen) aufzunehmen.

Abb. 3-9.: (* Struktur eines Datenbank-Anwendungssystems *)



Als Strukturierungsprinzip zum Aufbau der Transaktionen ist zu bemerken, daß möglichst kein Datentransfer von einem VADT zu einem anderen VADT über die Transaktion durch Parameterübergabe erfolgen sollte. Der Austausch von Daten zwischen VADT's sollte durch das Teilen gemeinsamer Objekte in der Datenbank geschehen. Die Transaktionen beschränken sich dann auf die Darstellung der Kontrollstrukturen zur Steuerung des Ablaufs.

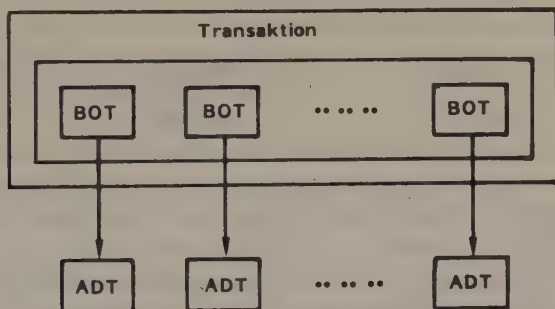
Zur Gewährleistung der Atomarität

Ohne operationale Sicht konnte mit Hilfe der Klammerung von BOT- und EOT-Operation unmittelbar die Atomarität einer Folge von Datenbank-Operationen gekennzeichnet und die notwendigen Vorkehrungen und Prüfungen eingeleitet werden. Den Anwendungsmoduln ist diese Klammerung einer Folge von Operationen transparent zu machen. Die vorgeschlagene Vorgehensweise orientiert sich an die Behandlung der BOT- und EOT-Behandlung in verteilten Datenbanksystemen (vgl. hierzu <Lin79>, <Gra80>).

Jeder Anwendungsmodul kennt die Operationen BOT und EOT. Die Benutzung eines Moduls durch eine Transaktion wird durch diese Operationen eingeleitet und abgeschlossen. Das BOT bzw. EOT einer Transaktion wird realisiert als ein BOT bzw. EOT Aufruf an alle in der IMPORT-Liste der Transaktion spezifizierten Moduln. In den Moduln kann dann eine modulspezifische BOT bzw. EOT Behandlung eingeleitet werden. Z.B. können modulspezifisch Integritätsbedingungen geprüft und ergänzende, integritätssichernde Datenbank-Operationen ausgeführt werden (vgl. Kap. 5).

Abb. 3-10.:

(* Modulspezifisches Anmelden einer Transaktion *)

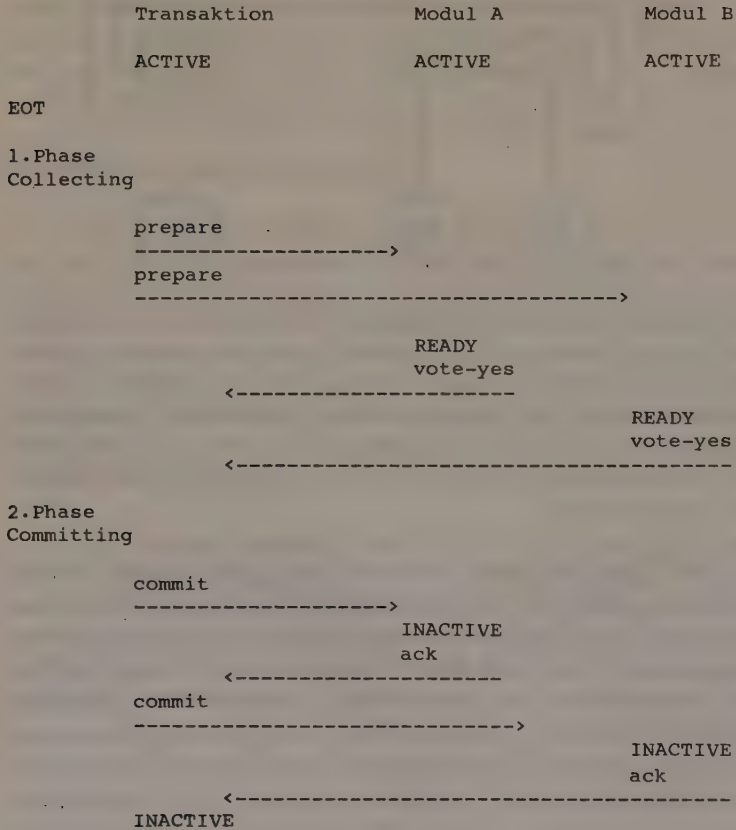


Der Abschluß der Transaktionen (als Commitment bezeichnet) hat nach dem Zwei-Phasen Commit-Protokoll (vgl. <Gra80>, <Lin79>) zu erfolgen, wenn die EOT-Behandlung modulspezifisch und in den Modulen von einander unabhängig erfolgt. Das Commit der Transaktion kann erst erfolgen, wenn alle Anwendungsmodulen ihre EOT-Behandlung ordnungsgemäß abgeschlossen haben.

Durch die EOT-Behandlung nach dem Zwei-Phasen Commit-Protokoll macht es keinen Unterschied, ob die Anwendungsmodulen auf verschiedenen Rechnern verteilt bzw. durch eigene Prozesse im Sinne des Betriebssystems realisiert sind. Wie die Überlegungen zu verteilten Datenbanken (<Gra80>, <Lin79>) zeigen.

Abb. 3-11.:

(* Zwei-Phasen Commitment einer Transaktion
nach <Lin79>, S.51 *)



3.4. Mögliche Konsequenzen für die Datenbank-Architektur

In den vorherigen Abschnitten wurde die Grundstruktur für den allgemeinen Aufbau von Datenbank-Anwendungssystemen entwickelt. Im generellen unterstützen Datenbanksysteme die Bildung von Datenbank-Aktionen nicht.

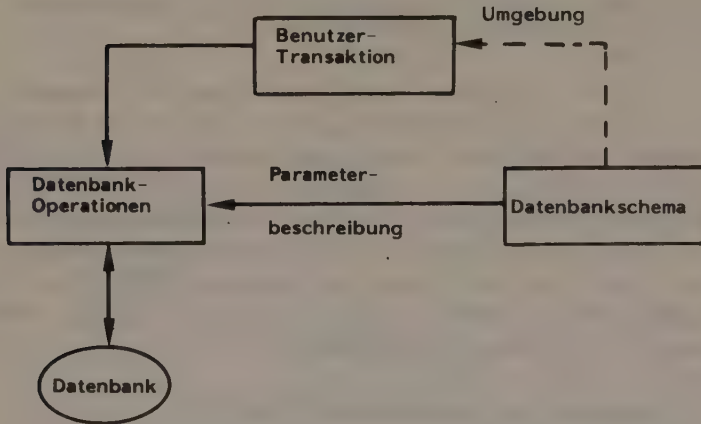
Es gibt allerdings zwei bemerkenswerte Ansätze zur Architektur von Datenbanksystemen, die die in den vorherigen Abschnitten gemachten Überlegungen teilweise berücksichtigen und die Basis für den Aufbau von Virtuellen Abstrakten Datentypen bilden können, da in deren Architekturvorstellungen eine entsprechende Integration einfacher einzubringen ist. Dies ist zum einen der Architekturvorschlag von Baroody/DeWitt <BDW81> und zum anderen der bereits erwähnte Ansatz von Rebsamen/Zehnder <RZ82>.

(1) Der Ansatz von Baroody/DeWitt <BDW81>

Zum Vergleich der Architekturen von Datenbanksystemen zeigt Abb. 3-12 den gängigen Aufbau, wobei die Transaktionen auf den Datenbank-Operationen aufsetzen. Die Datenbank-Operationen erhalten, notwendige Beschreibung über den Aufbau der Relationenschemata aus dem Konzeptionellen bzw. Externen Datenbankschema. Die Datenbankschemata bilden somit die Umgebung der Transaktion (oder des Benutzers). Über den Aufruf der Datenbank-Operationen wird der Datentransfer zwischen Datenbank und Transaktion realisiert.

Abb. 3-12.:

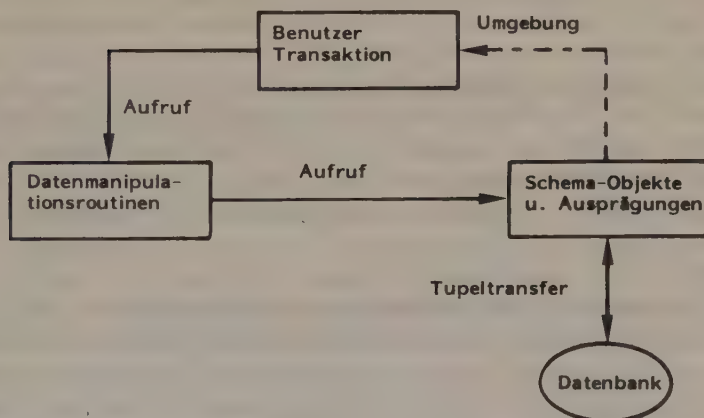
(* Datenbank-Architektur in gängiger Weise
nach <BDW81>, S.579 *)



Baroody/DeWitt fordern demgegenüber eine Festlegung der Objekte (schema objects) einer Datenbank in der Form Abstrakter Datentypen. Die Datenbankmanipulationsoperationen beziehen sich auf Ausprägungen dieser Objekte. Es ist deshalb bei der Ausführung einer Operation eine Zuordnung von den allgemeinen Datenbank-Operationen zu den spezifischen Ausprägungen eines Typ in der Datenbank zu treffen.

Abb. 3-13.:

(* Datenbank-Architektur in der von Baroody/DeWitt vorgeschlagener Form <BDW81>, S.585 *)



(2) Der Ansatz von Rebsamen/Zehnder <RZ82>

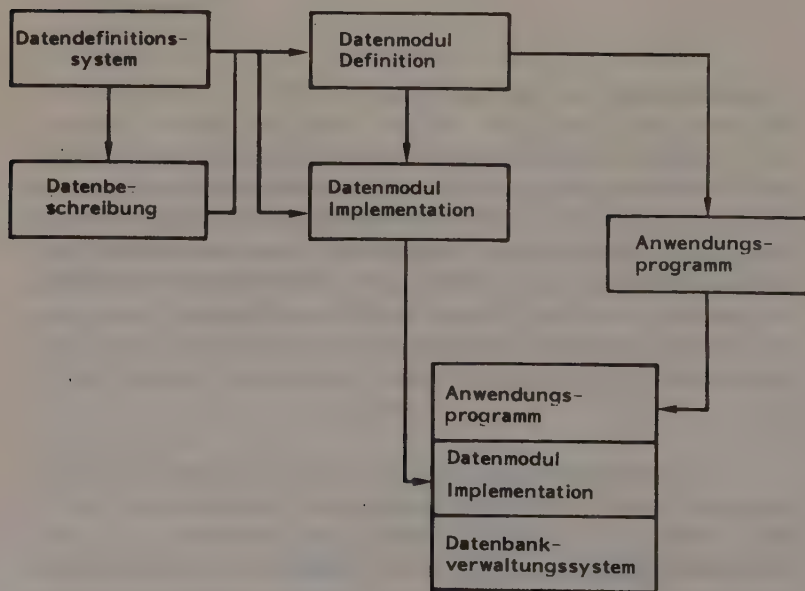
Der Ansatz von Rebsamen/Zehnder erscheint dem in Abs. 3.3 vorgetragenen Konzept ähnlich, dies liegt nicht zuletzt an der Verwendung der gleichen Wirtssprache, Modula-2, und der damit verbundenen Ähnlichkeit in der Datendefinition. Rebsamen/Zehnder bauen jedoch auf der Bereitstellung von Datenmoduln und deren Operationen als eine Menge konsistenz-erhaltender Transaktionen auf. Sie unterscheiden nicht zwischen Aktionen und Transaktionen im Sinne der Abschnitte 3.2 und 3.3.

Die Datenmoduln nach Rebsamen/Zehnder bilden einen Spezialfall des in Abs. 3.2 aufgezeigten Konzepts der VADT's. Ein Teilen gemeinsamer Objekte der Datenbank von verschiedenen Datenmoduln ist nicht vorgesehen. Die Möglichkeit zur Bildung von Sichten in der allgemeinen Form unterbleibt

damit. Die Operationen in den Datenmoduln (von Rebsamen/Zehnder als "konsistenzhaltende Transaktionen" bezeichnet) entsprechen den Aktionen eines VADT's. Die Konsistenzhaltung ist dabei sehr eingeschränkt zu verstehen, da sie den Konsistenzbegriff zugrunde legen, wie er von Thurnherr <Thu80> eingeführt wurde. Dieser umfaßt nur einen Teilaspekt der Datenbank-Integrität (vgl. dazu auch Kap. 5). Durch diese Vereinfachung können Aktionen im hier gebrauchten Sinne auch als Transaktion bezeichnet werden. Für die pragmatischen Aspekte der Transaktion, zur allgemeinen Darstellung der Handlungen einer Miniwelt, wie sie in Abs. 2.1 entwickelt wurden, erscheint dieses Vorgehen zu restriktiv.

Abb. 3-14.:

(* Erzeugung von Datenmoduln und deren Einbettung in das Anwendungsprogramm nach Rebsamen/Zehnder <RZ82>, S.604 *)



Der in Abs. 2.2.1 aufgezeigte Prozedurgedanke für die Darstellung von Transaktionen wurde von Rebsamen/Zehnder ihrem Ansatz zugrunde gelegt; der wie gezeigt, erweiterungsbedürftig ist. Der Ansatz weist jedoch den Weg auf im Hinblick auf Möglichkeiten zur Realisierung Virtueller Abstrakter Datentypen.

Möglichkeiten der Bildung einer Operationalen Sicht auf der Basis gängiger Datenbanksysteme

Versucht man auf gängigen Datenbankssystemen eine operationale Sicht zu realisieren, so ist eine entsprechende Wirtssprache erforderlich, die Modulstrukturen unterstützt. In diesem Fall fehlt allerdings die Möglichkeit, die Verwaltung der Anwendungsmoduln dem Datenbanksystem zu übertragen. Vom System ist es folglich nicht möglich, die Datendefinitionen der Moduln auf Konsistenz mit den Schemavereinbarungen aus der Datenbank zu prüfen. Für die Realisierung von Sicherungsmaßnahmen zur Datenbank-Integrität sind dies wesentliche Einschränkungen.

Andererseits bietet das aufgezeigte prinzipielle Vorgehen gewisse Vorteile in der Programmierung von Datenbank-Anwendungen. Dies wird in ersten Ansätzen durch die Arbeiten <Zör83>, <Fie83> bestätigt.

In gängigen Systemen (vgl. <SB83>) sind die Möglichkeiten zu Änderungen auf Sichten sehr eingeschränkt. Es fehlt eine explizite Möglichkeit der Übersetzung von Datenbank-Operationen. Es sind folglich nur einfache Sichtbildungsmechanismen zulässig, denen eine eindeutige Repräsentationsfunktion zugrunde liegt. Die Spezifikation der operationalen Sicht und die direkte Implementierung der Aktionen auf der Datenbank bietet hierzu eine Alternative (wie in <MS83>, <Fie83> geschehen).

Ein wichtiger Gesichtspunkt ist die Mächtigkeit der zur Verfügung stehenden Wirtssprache, die ebenfalls Mengenkonstrukte beinhalten sollte, um eine adäquate Möglichkeit zur Bearbeitung der Datenbank zu ermöglichen. Ansonsten sind verschiedene Anstrengungen zur Bereitstellung von Hilfskonstrukten zur Realisierung einer operationalen Sicht zu unternehmen.

Die Notwendigkeit eines Systems zur umfassenden Verwaltung eines Datenbank-Anwendungssystem einschließlich der operationalen Komponenten, d.h. eine Erweiterung des Data Dictionary auch um Aktionen, ist unabdingbar.

Eine alternative Struktur des Transaktionssystems

Eine alternative Struktur zu der in Abs. 3.3 angegebenen Realisierung von Transaktionen auf einer Menge von Datenbank-Anwendungsmoduln, ergibt sich durch eine ereignisgesteuerte Betrachtung. Die VADT's sind als Prozesse aufzufassen, die beim Erreichen bestimmter Datenbankzustände (Eintritt eines spezifischen Ereignisses) eine Folge von Aktionen ausführen.

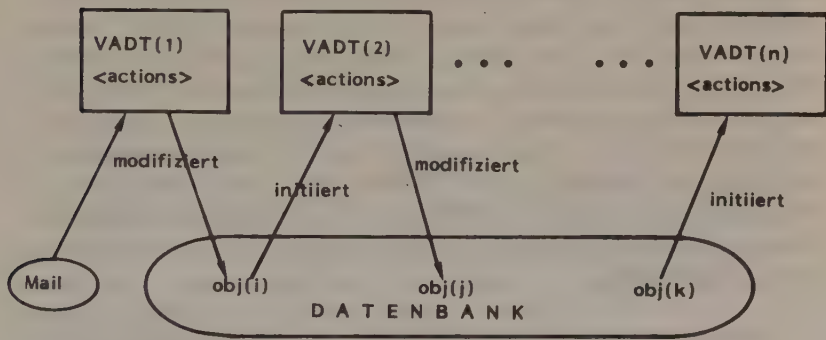
Die Realisierung einer Transaktion ergibt sich als eine Folge von Ereignissen, die verschiedene VADT's zu (Re-)Aktionen zwingen. Der Anstoß eines VADT's erfolgt durch eine von einem DC-Modul erzeugte Nachricht. Diese Nachricht bewirkt die Ausführung einer Folge von Aktionen durch den angesprochenen VADT. Durch den erzeugten Datenbankzustand wird wiederum ein VADT aktiviert, was eine Kettenreaktion von Aktivierungen von VADT's auslöst. (Die Kommunikation zwischen den VADT's wird durch den Austausch von Nachrichten als Zustände der Datenbank realisiert.)

Eine Transaktion (im Sinne der Darstellung einer Handlung der Miniwelt) manifestiert sich auch in diesem Fall als eine Folge von Zustandsübergängen in der Datenbank. Sie wer-

den jedoch nicht explizit in Form eines Programms angegeben, sondern entstehen durch das Zusammenwirken eigenständiger Abstrakter Datentypen.

Abb. 3-15.:

(* Realisierung der Transaktionen durch Nachrichten *)



Dieser Ansatz wirft allerdings einige Fragen hinsichtlich der Behandlung von Fehlerfällen auf, deren Erörterung den behandelten Rahmen sprengen würde.

4. Die Schematisierbarkeit der Transaktionen

- Basis für ein Transaktionssystem

Nachdem die Grundstrukturen von Transaktionen erörtert und ein erster Ansatz zum Aufbau eines transaktionsorientierten Datenbank-Anwendungssystems aufgezeigt wurde, können die Kriterien für die Schematisierbarkeit von Transaktionen behandelt werden. Es wird damit die Realisierbarkeit von Transaktionen im Sinne des Prozeduransatzes (vgl. Abs. 2.2.1) diskutiert. Das wesentliche Merkmal ist die Höhe des Abstraktionsniveaus der virtuellen Maschine, die dem Benutzer zur Verfügung gestellt werden kann. Im Hinblick auf die Automatisierbarkeit ist zu hinterfragen, inwieweit sich eine Handlung vollständig in eine Transaktion übertragen läßt. Die durch die Transaktionskriterien gesetzten Grenzen (Handlungen aus der Miniwelt als Transaktionen zu realisieren) sind zu beachten.

Wie im Abs. 2.2 erläutert wurde, ist das Konzept der "Transaktionsprozeduren" nicht hinreichend. In einem Datenbank-Anwendungssystem sind auch Transaktionen zu realisieren, die gegebenenfalls mit mehreren Benutzern und Transaktionen der Datenbank kooperieren, um so eine komplexe Handlung der Miniwelt zu realisieren.

Im folgenden Abschnitt werden durch eine erste grobe Klassifikation der Transaktionen nach der Art ihrer Schematisierbarkeit die Voraussetzungen zur Übertragbarkeit von Handlungen in ein Datenbank-Anwendungssystem skizziert. Die Schematisierbarkeit bestimmt wesentlich die Gestaltung der Transaktionen. Die Abschnitte 4.2 - 4.4 befassen sich, darauf aufbauend mit der Realisierung der einzelnen Transaktionsarten im Datenbank-Anwendungssystem.

4.1. Klassifikation der Transaktionen nach ihrer Schematisierbarkeit

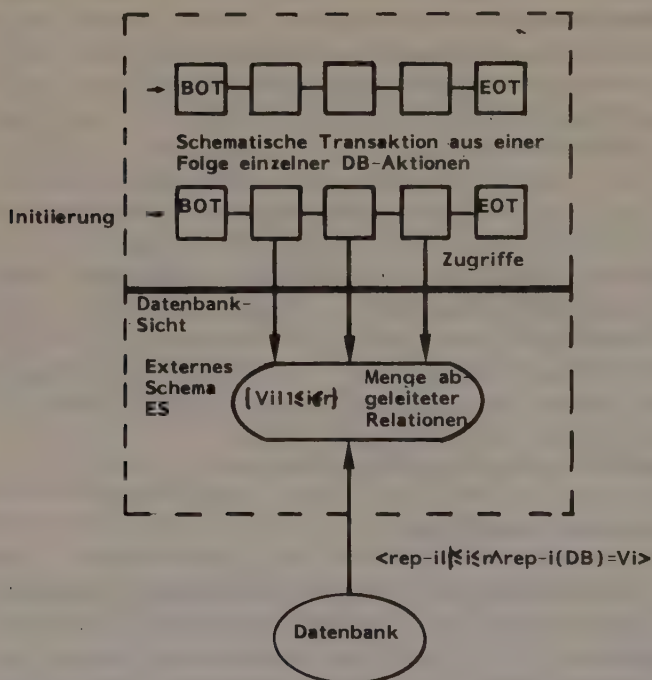
Im Abs. 2.2.2 wurde gezeigt, wie sich auf den elementaren Datenbank-Operationen anwendungsspezifische Operationen oder Aktionen bilden lassen. Der Übergang von einer Folge (für eine Schicht des Datenbank-Anwendungssystems) elementarer Operationen zu Operationen der im Abstraktionsniveau nächst höheren Ebene stellt eine funktionale Abstraktion dar. Es wird unter der Beibehaltung der Voraussetzungen und des Effekts vom Algorithmus abstrahiert. Der Abstraktion geht dabei eine "funktionale" Komposition voraus, die die einzelnen Teiloperationen zusammenfaßt. Die Grenzen einer solchen funktionalen Komposition im Hinblick auf die Bildung von Transaktionen (vgl. Abs. 2.2.1) wurden bereits skizziert. Eine Klassifikation der Transaktionen kann danach aufgebaut werden, inwieweit die Transaktionen aus zusammengesetzten Aktionen gebildet werden können. Es ergeben sich folgende Transaktionsarten:

(1) Schematische Transaktionen

Der Transaktionsablauf ist vollständig bestimmt. Zur Beschreibung der Transaktion könnte man eine Aktionsprozedur mit Transaktionseigenschaften angeben. Die Eingabe der Parameterwerte durch den Benutzer kann zu Beginn der Transaktion erfolgen. Es werden keine weiteren Eingaben erforderlich. Die Transaktion läuft aufgrund des gegebenen Zustands der Datenbank deterministisch ab. Die auf diese Weise realisierbaren Transaktionen entsprechen dem in Abs. 2.2.1 behandelten Fall. Durch die Transaktion wird ein funktionaler Übergang aus dem Zustand der Datenbank (vereinigt mit den Werten der Eingabeparameter) in einen Zustand der Datenbank (vereinigt mit den Werten der Ausgabeparameter) realisiert. Rebsamen/Zehnder <RZ82> bezeichnen diese Art der Transaktion auch als standardisierte Transaktionen.

Abb. 4-1.:

(* Struktur schematischer Transaktionen *)



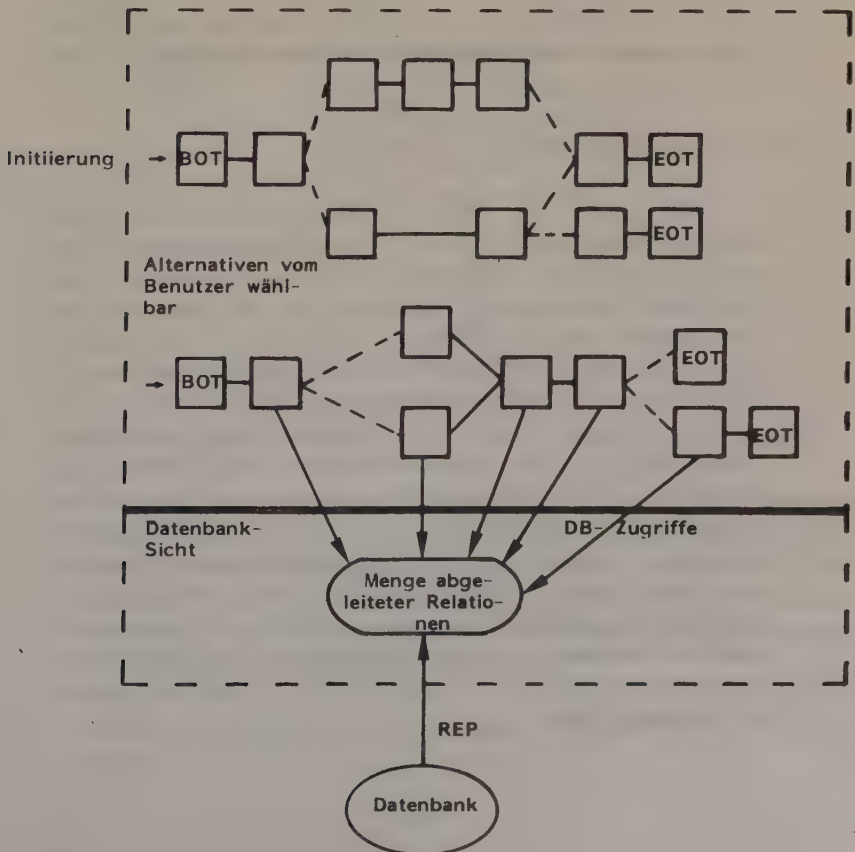
(2) Reglementierte Transaktionen

Der Ablauf der Transaktion ist zum Zeitpunkt der Initialisierung durch die Übergabe der (Anfangs-)Parameterwerte nicht vollständig bestimmt. Der Ablauf erfolgt vielmehr in Kooperation mit dem Benutzer und ist abhängig von Zwischenzuständen der Datenbank. Die Entscheidungen über den weiteren Verlauf der Transaktion werden durch den Benutzer direkt (durch Funktionsaufruf)

oder indirekt (durch Eingabe neuer Parameterwerte) getroffen. In Abs. 2.2.1 wurde hierfür bereits ein entsprechendes Beispiel skizziert. Der Ablauf der Transaktion ist nicht völlig willkürlich, sondern folgt bestimmten Regeln, die die Struktur der Transaktion festlegen. Die Transaktionen, die sich auf diese Weise darstellen lassen, werden im folgenden als reglementierte Transaktionen bezeichnet.

Abb. 4-2.:

(* Struktur reglementierter Transaktionen *)



Eine reglementierte Transaktion benutzt einen oder mehrere Anwendungsmoduln der Datenbank (wie sie im Abs. 3.2 eingeführt wurden). In diesen Moduln können durchaus sehr komplexe Aktionen realisiert sein. Eine reglementierte Transaktion besteht somit letztlich aus einer Folge funktionaler Übergänge, die aber insgesamt einen relationalen Übergang darstellen. Die der konkrete Anfangs- und Endzustand einer reglementierten Transaktion ist apriori nicht angebar. Es ist jedoch die Menge der möglichen Endzustände spezifizierbar. Über die Struktur reglementierten Transaktionen finden sich in der Datenbank-Literatur spärliche Anhaltspunkte, obwohl sie für die Behandlung der Integrität von Bedeutung sind. Einen ersten Ansatz zur Behandlung reglementierter Transaktionen kann man im "in-line-processing" von Davies <Dav78> sehen.

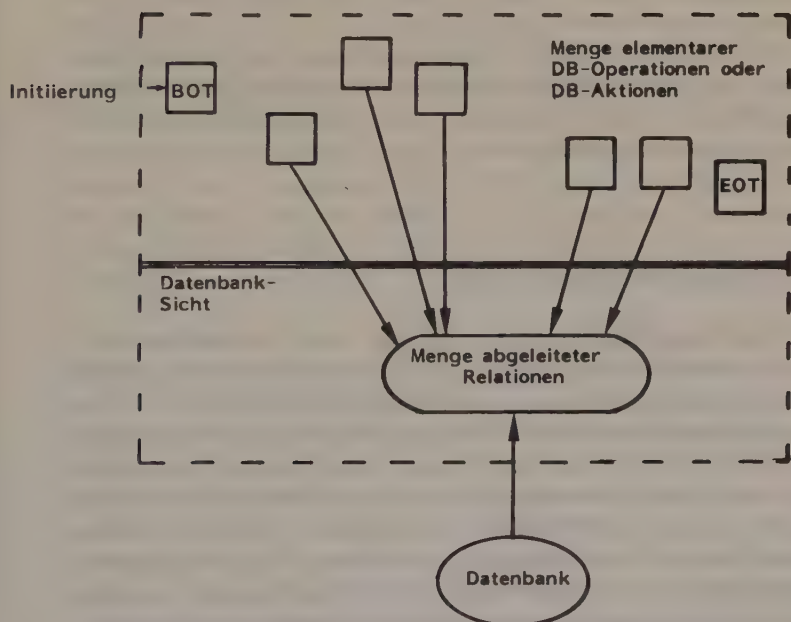
(3) Freie Transaktionen

Für die Art der freien Transaktionen gilt, daß der Transaktionsverlauf apriori nicht angebar ist. Durch den Benutzer der Transaktion werden eine Reihe von (Datenbank-)Operationen initiiert, die er letztlich als eine atomare Einheit aufgefaßt haben möchte. Der Benutzer kann dabei auf zwei Schnittstellen aufsetzen. Es steht ihm zum einen die operationale Sicht der Datenbank, realisiert durch die Datenbank-Anwendungsmoduln (vgl. Abs. 3.2), oder zum anderen die lokale Sicht der Datenbank eines Externen Schemas mit den elementaren Datenbank-Operationen (vgl. Abs. 3.1) zur Verfügung. Ein typisches Beispiel für freie Transaktionen sind die sogenannten "Ad-hoc" Abfragen <Cha80>. Für den Transaktionsablauf ergibt sich, daß jede Aktion auf die Datenbank vom Benutzer initiiert wird und aus der Sicht einer Transaktionsüberwachung keine Struktur in der Reihenfolge erkennbar wird.

Freie Transaktionen sind aus der Sicht der Datenbank relationale, nicht determinierte Übergänge.

Abb. 4-3.:

(* Struktur freier Transaktionen *)



Die eben dargelegte Dreiteilung umfaßt die typischen Fälle des Arbeitens mit einer Datenbank, wenn man sie unter dem Gesichtspunkt der Transaktionsbildung betrachtet. Die verschiedenen Fälle bedingen zwangsläufig wesentliche Unterschiede in den Anforderungen an die Transaktionsüberwachung, bezüglich der Integritätssicherung.

In den folgenden drei Abschnitten soll jeweils aufgezeigt werden,

- wie die Zurückführung der Transaktionsarten auf die Handlungstheorie,
 - wie die strukturierte Darstellung der einzelnen Transaktionsarten und deren Integration in ein Transaktionssystem,
- herauszuarbeiten ist.

Getrennt behandelt werden die unterschiedlichen Maßnahmen zur Integritätsüberwachung in Kapitel 5, die von diesem Konzept ausgehend in einem Datenbank-Anwendungssystem erforderlich und machbar sind.

4.2. Schematische Transaktionen

Schematische Transaktionen können auf "schematisches" Handeln zurückgeführt werden. Liegt eine gewisse Situation in der Miniwelt vor und wird in dieser Situation eine Handlung ausgelöst, so ist ihr weiterer Verlauf und ihre Wirkung determiniert. Läßt sich der Ablauf schematisieren, d.h., ist ein Algorithmus angebbbar, liegt eine schematische Handlung vor. Eben jene schematischen Handlungen entsprechen den schematischen Transaktionen. Sie ergeben einen funktionalen Übergang aus einer gegebenen Situation in eine andere. Wie in Abs. 2.2.1 angesprochen, sind diese schematischen Transaktionen prinzipiell in der Form einer "Transaktionsprozedur" realisierbar.

4.2.1. Rekonstruktion

Schematische Transaktionen haben ihren Ursprung in schematischen Handlungen. Die Voraussetzung einer schematischen Transaktion liegt im Vorhandensein einer entsprechenden schematischen Handlung in der Miniwelt. Schematisches Handeln bedingt, daß im Handlungsschema ein Handlungsmuster angegeben werden kann. Es bedeutet aber auch, daß in der Begründung der Handlung auf Handlungsschema verwiesen werden kann (vgl. <Schw79>, S.93 und Abs. 2.1.3). Die Ausführung der Handlung kann über das Handlungsschema gerechtfertigt werden, wenn entsprechende Kriterien zur Beurteilung der Voraussetzungen und der Wirkung der Handlung existieren. (Dieser Gesichtspunkt ist den Überlegungen zur Integrität zu beachten, Abs. 5.2.1)

Vom schematischen Handeln wird gefordert, daß eine Handlung, einmal ausgelöst, entsprechend ihrem Schema (vgl. Abs. 2.1.2) abläuft. Ihr Verlauf ist bei einer vorliegenden Ausgangssituation determiniert. Zu treffende Entscheidungen während des Ablaufs sind apriori zum Anfangszeitpunkt vorgegeben. Unwägbarkeiten "menschlicher" Willensfindung oder andere eventuell während des Ablaufs auftretende "Überraschungen" sind ausgeschlossen. Für die Darstellung im Datenbank-Anwendungssystem genügt es eine Transaktion bereitzustellen, die den einer schematischen Handlung entsprechenden funktionalen Übergang in der Datenbank realisiert.

Zusammenfassend kann eine schematische Handlung in eine schematische Transaktion dann übertragen werden, wenn

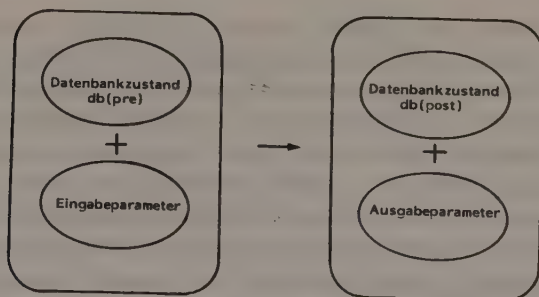
- (1) die Objekte der Handlung ebenfalls in der Datenbank repräsentierbar sind;
- (2) der funktionale Übergang bekannt ist, d.h. in Vor- und Nachbedingungen fixierbar ist;
- (3) das Handlungsmuster in prozeduraler Form angebbbar ist.

4.2.2. Strukturelle Darstellung

Eine schematische Transaktion beschreibt einen funktio-
nalen Übergang. Der Übergang vollzieht sich in der Form, daß
die zustandsbeschreibenden Parameter der Miniwelt entgegen-
genommen werden. Danach werden aufgrund der eingegebenen Para-
meterwerte und des aktuellen Zustands der Datenbank eine
Reihe von Aktionsprozeduren ausgeführt. Die Ausgabe der
Ergebnisse erfolgt am Ende der Transaktion. Die Folge von
Aktionen könnte grundsätzlich auch durch eine einzige Ak-
tionsprozedur mit Transaktionseigenschaften ersetzt werden.

Abb. 4-4.:

(* funktionaler Übergang *)

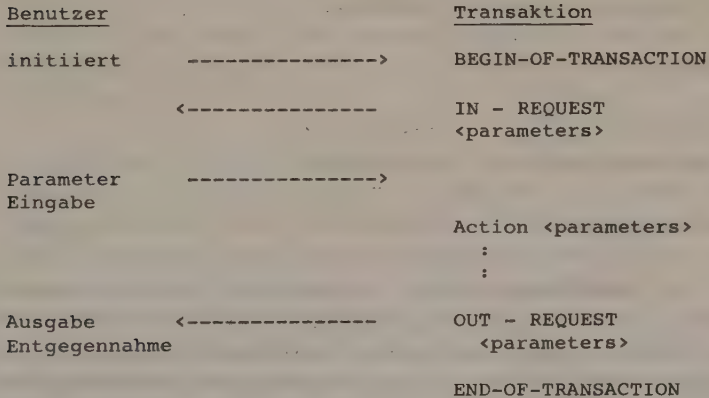


Die Ausführung der Folge der einzelnen Aktionsproze-
duren ist dabei für die Datenbank insgesamt konsistenz-erhal-
tend. Die Erhaltung der Integrität auch der Miniwelt, in der
durch die Ausgabe von Ergebnissen ebenfalls eine nicht
zulässige Situation entstehen kann, wird noch im Rahmen der
Arbeit nicht behandelt. Die ist die Aufgabe eines Sicher-
heitskontrollsystems, das in einer Output-Kontrolle den
Informationsfluß zum Benutzer überwacht (vgl. <Schl82>).

Schematische Transaktionen lassen sich bezüglich der Interaktion mit dem Benutzer auf folgendes, einfaches Schema zurückführen:

Abb. 4-5.:

(* Benutzer - Transaktion *)



Die "IN-" und "OUT-Requests gehen über einen DC-Modul zur Kommunikation mit dem Benutzer (vgl. Abs. 3.3).

Zur Spezifikation

Für die Spezifikation dieses Spezialfalles einer Transaktion ergibt sich in Anlehnung an die Notation von Moduln in Modula-2 (vgl. Abs. 3.3 und <RZ82>):

```
TRANSACTION transaction;
SPECIFICATION;

SCOPE;
FROM <db-application-module> IMPORT <actions>;
      :
FROM <dc-module> IMPORT <actions>;

CONDITIONS;

PRE-CONDITION:      <predicate>;
POST-CONDITION:     <predicate>;

END transaction.
```

Die Vorbedingung einer Transaktion, die als Voraussetzung der Aktualisierung erfüllt sein muß, besteht aus zwei Teilen, den invarianten Bedingungen bezüglich der Datenbank und den eigentlichen transaktionsspezifischen Vorbedingungen. Die invarianten Bedingungen sind Integritätsbedingungen der Datenbank. Sie werden in Abs. 5.3 noch ausführlich behandelt. Die transaktionsspezifischen Vorbedingungen lassen sich weiter unterteilen in Bedingungen bezüglich des Datenbankzustands und der Werte der Eingabeparameter.

Die Nachbedingung beschreibt, die Wirkung der Transaktion auf der Datenbank und bestimmt die Ausgabeparameterwerte. Die Nachbedingung ist als Implikation darstellbar, die von den Eingabeparameterwerten und dem Datenbankzustand ausgehend einen Zustand der Datenbank und die Ausgabeparameterwerte festlegt. (Die Nachbedingung kann aus mehreren Teilbedingungen (vgl. Abs. 4.3.2) bestehen, wichtig ist jedoch im Fall der schematischen Transaktionen, daß der Effekt beim Beginn der Transaktion in einem gegebenen Anfangszustand durch die Nachbedingung eindeutig festgelegt

ist.) Die invarianten Bedingungen der Datenbank müssen ebenfalls gelten, sie sind deshalb konjunktiv mit der transaktionspezifischen Nachbedingung zu verknüpfen. Während der Transaktionsausführung können sie jedoch verletzt sein.

Auf eine explizite Angabe der invarianten Bedingungen in der Spezifikation von Transaktionen kann verzichtet werden. Sie lassen sich günstiger Weise datenbankspezifisch notieren (vgl. Abs. 5.3). Für die Verifikation und die EOT-Behandlung der Transaktionen sind sie jedoch selbstverständlich erforderlich.

Die Spezifikation erfordert die Angabe der Prädikate in den Vor- und Nachbedingungen in der Form bool'scher Funktionen. Diese sind von den VADT'en, auf denen die Transaktion realisiert wird, bereitzustellen. Von dieser Vorgehensweise wird im Rahmen der Arbeit abgewichen. Es wird auf die nächsttiefer Schicht, nämlich auf die Datenmanipulationssprache bezüglich der Datenbanksicht, zurückgegriffen. (Durch die Formulierung der Vor- und Nachbedingungen in QUEL entfällt in den Beispielen des Anhangs 2 die explizite Angabe komplexer Leseoperationen und bool'scher Testfunktionen in den Datenbank-Anwendungsmoduln.) Dies bedeutet keine Einschränkung der Überlegungen, vereinfacht jedoch die Darstellung der Beispiele.

Der Benutzer initiiert die Transaktionen des ihm zugeordneten Bereichs des Datenbank-Anwendungssystems. Die Transaktion sorgt in ihrer Implementation zuerst für einen konsistenten Anfangszustand, durch die Ausführung einer BOT-Operation (vgl. Abs. 5.5.1). Die Übernahme der Eingabeparameterwerte (als Zustandsbeschreibung der Miniwelt) erfolgt über einen DC-Modul als Kommunikationsmodul zum Anwender. Die Entgegennahme kann durch mehrere Eingabeanforderungen erfolgen. Die eigentliche Bearbeitung erfolgt durch die Ausführung einer Folge von Aktionsprozeduren. Der Ablauf ist determiniert und dementsprechend durch Kontrollstrukturen

festgelegt. Mögliche Alternativen in der Bearbeitung werden bereits durch die Angabe entsprechender alternativer Parameterwerte apriori berücksichtigt. Nach der Ausgabe der Ergebnisse schließt die Transaktion im Rahmen der EOT-Behandlung mit der Herstellung eines konsistenten Zustands ab (vgl. Abs. 3.3 und Kap. 5).

Die schematischen Transaktionen umfassen genau die Klasse von Transaktionen, die im Ansatz von Rebsamen/Zehnder <RZ82> und Reimer/Schmidt <RS81> beschrieben werden. Durch das Ausklammern der DC-Aktionen lassen sie sich mit Hilfe von Aktionsprozeduren (vgl. Abs. 2.2.1) realisieren.

Ein einfaches Beispiel zur Spezifikation einer schematischen Transaktion findet sich im Anhang 4 als "statistischen Monatsbericht eines Pharmareferenten" für die Pharma-
außendienstleitung.

Zur Implementierung

Die Implementationstruktur dieses Spezialfalles wird hier nur skizziert, da sie im Aufbau den Aktionen (vgl. Abs. 2.2.2) entspricht. In Anlehnung an Modula-2 ergibt sich:

```
TRANSACTION transaction;  
IMPLEMENTATION;  
  
SCOPE;  
FROM <db-application-module> IMPORT ....  
:  
FROM <dc-module> IMPORT ....
```



```
BEGIN-OF-TRANSACTION;
  IN <parameters>;
  (* Steht für die Zusammenfassung der Parametereingaben *)
  IF NOT <pre-condition> THEN UNDO;
    :
    ACTION <parameters>;
    (* importiert aus den Datenbank-Anwendungsmoduln *)
    :
  IF NOT <post-condition> THEN UNDO;
  (* Überflüssig, wenn die Korrektheit der Implementation
    nachgewiesen wurde *)
  OUT <parameters>;
  (* Steht für die Zusammenfassung der Parameterausgaben *)
END-OF-TRANSACTION;
END transaction.
```

Auf das Prüfen der transaktionsspezifischen Nachbedingung kann verzichtet werden, wenn die Implementation bezüglich der Spezifikation als korrekt nachgewiesen wurde. Dies gilt auch für die invarianten Bedingungen der Datenbank, wenn man sie in die Verifikation der Transaktion mit einbezieht. (Im Rahmen der Integritätskontrolle (Abs. 5.5.1) werden Mechanismen erörtert, die es ermöglichen bei Verletzungen invarianter Bedingungen Aktionen zu aktivieren. Das Ziel dieser Aktionen ist es, den inkorrekten Zustand zu einem korrekten Zustand zu ergänzen.)

Schematische Transaktionen entsprechen in der Datenbankprogrammierung (vgl. <Dav78>) den klassischen "batch processing transactions" und teilweise den "on-line processing transactions" im Sinne eines Pseudo-Dialogs, indem nur die Paramterversorgung von Terminal aus erfolgt. Vom Benutzer wird ansonsten jedoch keine weitere Steuerung des Programmablaufs vorgenommen.

Einbettung ins Transaktionssystem

Die Einbettung schematischer Transaktionen in ein Transaktionssystem ist problemlos. Letztlich ist dies der einfachste Fall einer Transaktionsprogrammierung, der um integritätssichernde Komponenten zu erweitern ist. Zusammengefaßt zu lokalen Anwendungssystemen bilden die Transaktionen die Einheiten, über die die Initiierung und eine erste Zugriffsschutzkontrolle erfolgt (vgl. Abs. 6.2). Im Rahmen eines umfassenden "Data Dictionary"-Systems ist die Verwaltung der Transaktionen in dieses zu integrieren. Im Anhang 4 wird z.B. ein Ausschnitt des Transaktionssystems für die Pharmaaußendienstleitung angegeben, der auch die Realisierung einer schematische Transaktion enthält.

4.3. Reglementierte Transaktionen

Durch das Eingreifen des Benutzers in die Transaktionsausführung ist die Möglichkeit eines nicht funktionalen Übergangs von einem Zustand der Datenbank in einen anderen gegeben. Bei den Überlegungen zum Aufbau eines Datenbank-Anwendungssystems ist dies zu berücksichtigen. Das Eingreifen des Benutzers ist erforderlich, wenn Entscheidungen während der Ausführung einer Transaktion über ihren weiteren Ablauf zu treffen sind. Transaktionen dieser Form lassen sich nicht mehr durch einen rein schematischen Ablauf bezogen auf die Datenbank realisieren, wie er in Abs. 4.2 aufgezeigt wurde. Es ist in diesem Fall notwendig, eine Folge komplexer Aktionen auszuführen, deren Abfolge apriori nicht determiniert ist. Insgesamt soll diese Folge jedoch als atomar bezüglich der Datenbank sein. Die Notwendigkeit eines solchen Vorgehens läßt sich aus der Handlungstheorie heraus begründen. Es gilt eine mögliche Form zur Spezifikation solcher Transaktionen zu finden und deren Integration in das Anwendungs-Gesamtsystem und Möglichkeiten zur Implementierung aufzuzeigen.

4.3.1. Rekonstruktion

Reglementierte Transaktionen sind auf zwei Arten von Handlungen zurückzuführen. Wie schematische Transaktionen haben auch reglementierte Transaktionen zum einen ihren Ursprung im schematischen Handeln. Das Muster einer schematischen Handlung besteht aus einer Folge von Teilhandlungen. In den Zwischenzuständen einer konkreten Aktualisierung wird abhängig von Zuständen der Datenbank oder Parameterwerten auf verschiedene alternative Teilhandlungen verzweigt.

Wie von Wright in seinem Entwurf zur Handlungslogik (<Wri77>, S.83ff) dargestellt hat, liegt einer solchen Handlung ein Handlungsbaum zugrunde. Ausgehend von der konkret vorliegenden Situation wird bei einer Aktualisierung der Handlung ein Pfad in diesem Baumes durchlaufen.

Abb. 4-6.: (* Handlungsbaum *)



Die Darstellung einer solchen Handlung als schematische Transaktion würde voraussetzen, daß bei der Aktualisierung der Transaktion die Auswahl der Alternativen in den einzelnen "Knoten" bereits erfolgt ist. Hierzu muß die Ausgangssituation, in der die Transaktion aktualisiert wird,

vollständig bekannt sein. Dies ist jedoch nicht der Fall, wenn der Ablauf der Transaktion nicht vollständig durch Eigenschaften von Objekten der Datenbank bestimmt wird. Der Benutzer hat bei einer Realisierung solcher Handlungen mit alternativen Verläufen, die zur Auswahl notwendige Information der Transaktion zu übergeben. Da ihm wiederum der Zustand der Datenbank nicht bekannt ist, hat er für alle Verzweigungen im Handlungsbaum die entsprechenden Parameterwerte zur Auswahl bereitzustellen. Ermöglicht man die Eingabe der benötigten Information erst zum Zeitpunkt des Erreichens eines Zwischenzustands, erhält man eine reglementierte Transaktion. Die Wirkung dieser reglementierten Transaktion auf die Datenbank erscheint zum Zeitpunkt ihrer Aktualisierung aus der Sicht des Datenbank-Anwendungssystems als nicht deterministisch.

Neben den schematisierten Handlungen werden in der Handlungstheorie auch Handlungen betrachtet, die bei ihrer Aktualisierung in ihrem Ablauf noch nicht determiniert sind. Schwemmer <Schw79> bezeichnet diese Handlungen als ein Handeln in Regeln. Einem Handelnden ist es freigestellt, in bestimmten Situationen aus einer Menge von Handlungen zu aktualisieren oder zu unterlassen. Welche Handlung er auswählt, entscheidet er rein subjektiv. Festgelegt sind die ihm zur Verfügung stehenden Alternativen und der Zweck, den sein Handeln verfolgt.

Dieses Handeln nach Regeln ist durch schematische Transaktionen nicht mehr darstellbar, wenn man eine zweckgerichtete Folge solcher reglementierter Teilhandlungen als integritätserhaltende Einheit in einem Datenbank-Anwendungssystem realisieren möchte.

Ein adäquates Mittel zur Realisierung des Handelns nach Regeln ist die erwähnte Form der reglementierten Transaktion. Der Benutzer initiiert eine Transaktion. In den Zwischenzuständen, in denen verschiedene Alternativen von Teil-

handlungen möglich sind, entscheidet der Benutzer, welche weitere Aktion er aktualisiert. Eine reglementierte Transaktion besteht somit aus einer Kette von Teilhandlungen.

Zusammenfassend gilt, Handlungen sind als reglementierte Transaktionen zu realisieren,

- (1) wenn Entscheidungen vom Benutzer während der Ausführung Transaktion zu treffen sind,
- (2) wenn der Zustand der Miniwelt in eine "Vielzahl" von Entscheidungen eingeht.

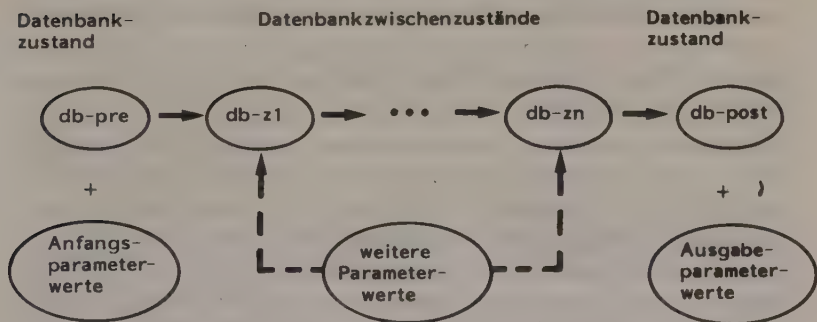
Reglementierte Transaktionen sind nur im direkten Dialog mit dem Benutzer zu realisieren. Da sie sich aus durchaus komplexen Teilhandlungen zusammensetzen, sind sie einer Überwachung zugänglicher als die freie Benutzung einer Datenmanipulationssprache.

4.3.2. Strukturelle Darstellung

Wie aus den Überlegungen des vorherigen Abschnitt zu erkennen ist, sind die reglementierten Transaktionen intentional bestimmt. Die intendierten Situationen (als eine Menge möglicher Datenbankzustände) werden in den Nachbedingungen der Transaktion fixiert. Dagegen ist der Weg, d.h. die Auswahl der notwendigen Aktionsfolgen, noch offen bzw. nur in Teilsequenzen festgelegt. Aber es existiert eine Folge von Aktionen, die in einer zu findenden Kombination die gewünschte Situation erzeugt.

Abb. 4-7.:

(* relationaler Übergang als Folge funktionaler Teilübergänge *)



Verschiedene Arten von Parametern

Der Mangel an Kenntnis über den exakten Verlauf der Transaktion hat letztlich zwei sich auf den Zustand der Miniwelt beziehende Ursachen, wie in Abs. 4.3.1 gezeigt. Sie spiegeln sich in der Art der durch den Benutzer eingegebenen Parameter wider.

Es lassen sich zwei Arten von Parametern unterscheiden,

- Werte zur Beschreibung eines Zustands der Miniwelt, die zur Übernahme in die Datenbank oder zur Auswertung der Datenbank notwendig sind, und
- Werte ohne direkten Einfluß auf die Datenbank, die ausschließlich zur weiteren Steuerung Transaktionsablaufs erforderlich sind.

(1) Eingabe von Zustandsbeschreibungen aus der Miniwelt

Der Ablauf der Gesamthandlung (bestehend aus einer Teilhandlung in der Miniwelt und der Transaktion) ist determiniert bezüglich des verfolgten Ziels. Nur die benötigten Zustandsbeschreibungen sind sowohl auf die Datenbank als auch auf die Miniwelt verteilt. (Sie sind somit vom Benutzer einzugeben.) Die Transaktion ist deshalb auf die Kommunikation mit dem Benutzer angewiesen. Die Eingabe aller möglichen Parameter für die verschiedenen Alternativen im Ablauf der Transaktion zum Zeitpunkt der Initialisierung wäre im Prinzip möglich. Man greift hier jedoch in der Regel auf den Dialog zwischen Benutzer und Transaktion zurück und versorgt die Parameter nur, wenn sie aktuell benötigt werden.

Transaktionen, die ausschließlich Dialogeingaben von Parametern dieser Art besitzen, sind streng reglementiert.

(2) Eingabe von Parametern zur direkten Gestaltung des Transaktionsablaufs

Sind in einer Situation die Folgeaktionen nicht bestimmt, d.h. sind sie "frei" wählbar, dann ist es erforderlich, dem Benutzer die Auswahl zu überlassen. Dies ist insbesondere der Fall bei nicht deterministischen Entscheidungen, die dem subjektiven Willen des Benutzers unterliegen. Sind die Kriterien für die Beurteilung einer gegebenen Situation unklar, legt der Benutzer konkret fest, wie am "günstigsten" zu handeln ist. (Ein erläuterndes Beispiel hierfür findet sich im Anhang 3). Die Auswahl der Aktionen ist dabei ebenfalls nicht schematisierbar. Diese lose Reglementierung kann bis zur vollständigen Freiheit der Auswahl einer Folge von Aktionen aus einer spezifizierten Menge gehen, wenn die Entscheidungskriterien für die Auswahl der Aktionen nicht oder nur aufwendig formalisierbar und damit programmierbar sind. Transaktionen dieser Art sind nur lose reglementierbar.

Streng und lose reglementierte Transaktionen unterscheiden sich somit aus der Sicht des dargestellten relationalen Übergangs.

Streng reglementiert:

Relationaler Übergang aus der Sicht der Datenbank, jedoch funktionaler Übergang aus der Sicht der Miniwelt und der Datenbank als Ganzes.

Lose reglementiert:

Relationaler Übergang auch aus der Sicht von Datenbank und Miniwelt, abhängig vom subjektiven Willen des Benutzers.

streng reglementierte Transaktionen

Zuerst werden jene Transaktionen, die eine Einbeziehung des Benutzers ausschließlich zur Zustandsaufnahme der Miniwelt benötigen, ausgegrenzt. Eine Transaktion dieser Art kann in üblicher Weise spezifiziert und durch ein, wenn auch eventuell komplexes, Anwendungsprogramm implementiert werden. Soweit Zustände der Miniwelt betroffen sind, erfolgt die Versorgung der Parameter der Datenbank-Aktionen jeweils aktuell durch den Benutzer. Bei der Versorgung aller Parameter zu Beginn einer reglementierten Transaktion wäre es notwendig, bereits alle möglichen alternativen Abläufe zu bedenken. Die Problematik liegt darin, daß gerade aufgrund der Zugriffe auf die Datenbank im Verlauf der Transaktion dem Benutzer bestimmte Situationen bekannt werden, auf die er deterministisch reagiert. Die anderen, apriori möglichen Alternativen in einem konkreten Ausgangszustand der Datenbank jedoch irrelevant sind.

Beispiele für streng-reglementierte Transaktionen finden sich im Anhang 3. Die Implementation einer streng reglementierbaren Transaktion ist der klassische Fall eines Dialogprogramms.

lose reglementierte Transaktionen

Im Transaktionsprogramm bedeutet dies, daß die Bedingungsvariablen in den Kontrollstrukturen erst bei Bedarf durch den Benutzer versorgt werden. Es werden immer dann, wenn eine Bedingungen im Transaktionsablauf zur Entscheidung ansteht, den noch nicht belegten Variablen durch "Kommunikation" mit dem kooperierenden Benutzer in "sinnvoller" Weise Werte zugewiesen. Die lose reglementierte Transaktion minimiert somit auch die Parametereingaben bezüglich der Steuerung des Ablaufs.

Das entscheidende Kriterium zur Festlegung einer losen reglementierten Transaktion ist die Nachbedingung. Sie bestimmt letztlich, ob eine Folge ausgeführter, benutzer-initiiertter Aktionen eine zulässige Transaktion bildet.

Die Folge der Aktionen läßt sich innerhalb einer Transaktion schrittweise weiter einengen. Man könnte die Gültigkeit von Teilbedingungen in Zwischenzuständen fordern (also "lokale" Nachbedingungen einführen). Es wären damit nur bestimmte "Pfade" von Aktionen, auf denen die Transaktion voranschreiten kann, zulässig.

(Die Einengung der in einer Transaktion möglichen Aktionen zeigt starke Parallelen zu Mechanismen der Synchronisation von Operationen in Abstrakten Datentypen. Auch innerhalb der Behandlung von Synchronisationsproblemen kann man mit zustandsbeschreibenden Prädikaten die Folge der Ausführbarkeit von Operationen beschränken (vgl. hierzu insbesondere <Ker82>, S.32).

Streng reglementierte Transaktionen können als ein spezieller Fall der lose reglementierten aufgefaßt werden. Es entfällt die Auswahl von Aktionen durch den Benutzer. Sie ist ersetzt durch Bedingungen über Zustandsvariablen, die durch die Datenbank oder Parameterwerte versorgt werden.

Zur Spezifikation

Die Spezifikation einer reglementierten Transaktion unterscheidet sich in der Vorbedingung über den Anfangszustand nicht von einer schematischen Transaktion. Die Nachbedingung jedoch spezifiziert eine Menge möglicher, alternativer Effekte der Transaktion. Der durch eine konkreten Ausprägung der Transaktion erzeugten Wirkung entspricht einer dieser spezifizierten Effekte. Die Nachbedingung einer reglementierten Transaktion ergibt sich als disjunktive Verknüpfung zustandsbeschreibender Prädikate.

```
TRANSACTION transaction;  
SPECIFICATION;  
SCOPE;  
FROM db-application-module IMPORT <actions>;  
:  
FROM dc-module IMPORT <actions>;  
  
CONDITIONS;  
PRE-CONDITION:          <predicate>;  
  
POST-CONDITION:         <predicate> OR .... OR <predicate>;  
END transaction.
```

Durch jedes Teilprädikat in der Nachbedingung wird ein möglicher Übergang spezifiziert. Die Übergänge sind als verschiedene Alternativen aufzufassen. Welche der Alternativen bei einer Ausführung der Transaktion realisiert wird, ist erst während der Ausführung entscheidbar.

Wie bereits bei den schematischen Transaktionen kann auch hier auf die explizite Angabe der invarianten Bedingungen bezüglich der Datenbank verzichtet werden. Sie gehören ebenfalls implizit zur Spezifikation und sind bei der einer Verifikation oder Validierung der Implementation zu berücksichtigen.

Die Nachbedingung der regementierten Transaktion umfaßt wiederum Bedingungen über die Ausgabeparamterwerte sowie über den Zustand der Datenbank. In der angegebenen Form ist in der Spezifikation loser und streng reglementierter Transaktionen kein Unterschied zu erkennen. Näher betrachtet, unterscheiden sich streng und lose reglementierte Transaktionen wesentlich von einander durch die Struktur der einzelnen, disjunktiv verknüpften Teilprädikate der Nachbedingung.

Bei streng reglementierten Transaktionen beschreiben die einzelnen Teilprädikate die Wirkungen in Abhängigkeit von den Eingabeparameterwerten und dem Zustand der Datenbank vor dem Zeitpunkt der Transaktionsausführung. Es handelt sich bei den Teilprädikaten um Implikationen, der Form "Prädikat(vorher) $\rightarrow$ Prädikat(nachher)". Die Prädikate sind in den einzelnen Teilprädikaten über den Zustand vor der Transaktionsausführung disjunkt.

Sei $NB = (TP_1 \text{ OR } \dots \text{ OR } TP_n)$ Nachbedingung mit $TP_i = (VP_i \rightarrow NP_i)$ einer streng reglementierten Transaktion T so gilt:

$$\bigwedge_j (j \neq i \text{ AND } VP_i \rightarrow \text{NOT } VP_j)$$

Bei streng reglementierten Transaktionen handelt es sich somit um einen funktionalen Übergang dessen Vorbedingungen zum Zeitpunkt der Initialisierung nicht (bzw. nur unter großem Aufwand) prüfbar sind. Vom Zeitpunkt des Transaktionsbeginns aus betrachtet erscheint deshalb der Übergang bezüglich der Datenbank als relational.

Die Teilprädikate in der Nachbedingung lose reglementierter Transaktionen sind nicht vom obigen Typ. Soweit die Teilprädikate als Implikationen dargestellt werden, ist der erzeugte Zustand nicht direkt abhängig vom Zustand vor der Ausführung der Transaktion. Die Wirkung der Transaktion wird durch Prädikate über Variable mit bestimmt, die den Verlauf der Transaktion steuern. Der Zustand dieser Variablen ist indeterministisch, da er direkt und während der Ausführung vom Benutzer manipuliert wird. (Man vergleiche die zuvor gemachten Aussagen über die zwei Arten von Parametern). Die Teilprädikate für lose reglementierte Transaktionen könnte man in die folgende Form bringen. SP_i steht dabei für ein Prädikat über die Eingabeparameter, die erst während der Transaktionsausführung belegt werden.

Sei $NB = (TP_1 \text{ OR } \dots \text{ OR } TP_n)$ Nachbedingung mit $TP_i = (VP_i \text{ AND } SP_i \text{ ---> } NP_i)$ einer lose reglementierten Transaktion T , so gilt nach Ausführung der Transaktion:

$$\bigwedge_j (j \neq i \text{ AND } VP_i \text{ AND } SP_i \text{ ---> NOT } (VP_j \text{ AND } SP_j))$$

Vor Ausführung der Transaktion ist jedoch der Wahrheitswert der Prädikate SP_j indifferent.

Die Spezifikation lose reglementierter Transaktionen kann in zwei Schritten erfolgen.

In einem ersten Schritt kann auf die Angabe der Prädikate des Typs SP_i in den Teilprädikaten verzichtet werden. Sie beziehen sich auf Variable, die nur den Ablauf der Transaktion bestimmen. Die Belegung dieser Variablen sagt nichts aus über die Möglichkeiten, aus einem gegebenen Datenbankzustand eine Menge neuer Zustände zu erzeugen. (Für die Verifikation der Implementation lose reglementierter Transaktionen sind die Prädikate vom Typ SP_j sehr wohl von Interesse. Vom Aspekt der Struktur und der anwendungsbezogenen Semantik der Transaktion als Repräsentation möglicher Handlungen sind sie irrelevant.)

In einem zweiten Schritt sind die Teilprädikate in der Nachbedingung um Prädikate des Typs SP_i zu ergänzen. Allerdings wird damit auch die Abfolge (d.h. der Handlungsbaum) der Aktionen innerhalb der Transaktion näher bestimmt.

Es wird aus der Spezifikation noch einmal ersichtlich, daß im Falle der reglementierten Transaktionen kein funktionaler Übergang bezüglich der Datenbankzustände vorliegt. Beispiele für reglementierte Transaktionen finden sich im Anhang 3 und 4.

Bei beiden Arten der reglementierten Transaktionen handelt es sich im Sinne von Davies <Dav78> um ein sogenanntes "in-line processing". Der Benutzer wird selbst zum Bestandteil der Transaktion, der Transaktionsablauf wird also von

ihm direkt mitbestimmt.

4.3.3. Möglichkeiten zur Implementierung

Der folgende Rahmen zur Struktur der Implementation reglementierter Transaktionen basiert auf der Angabe von Vorbedingungen für die einzelnen Aktionen. Es werden damit die möglichen Abläufe schrittweise fixiert. Die Vorgehensweise orientiert sich an die Strukturierung von Dialogprogrammen, wie sie von Brown <Bro82> vorgeschlagen wird. Sie ist in gewisser Weise aktionsmengenorientiert. Eine Menge wechselseitig abhängiger Aktionen wird durch gemeinsame Vorbedingungen zusammengefaßt. Durch Funktionsauswahl oder durch Eingabe entsprechender Parameterwerte wird vom Benutzer eine Aktion initiiert.

In dem Ansatz von Brown <Bro82> *) wird deutlich, daß für die Darstellung des Verlaufs im wesentlichen die aus der Programmierung bekannten Kontrollstrukturen auftreten. Die Bedingungen in den Kontrollstrukturen zur Auswahl einer Aktion sind direkt abhängig von den Eingaben des Benutzers.

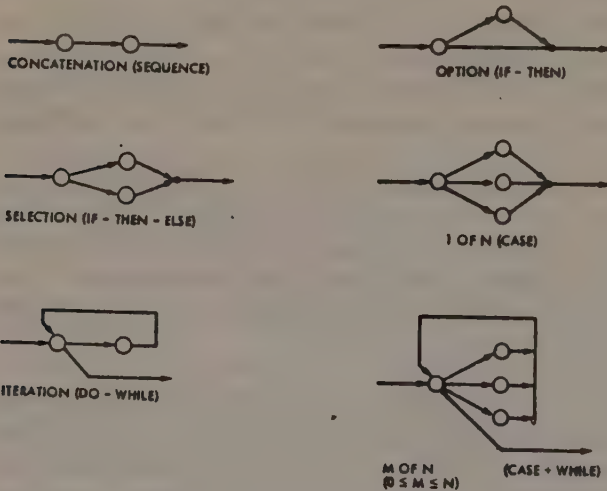
Entsprechend zu Brown (<Bro82>, S.413) sind die einzelnen Elemente in Abb. 4-8 als Teile eines gerichteten Graphen aufgeführt. Die Knoten in Abb. 4-8 kennzeichnen, daß eine Eingabe vom Benutzer erforderlich ist. Die Pfeile stehen für eine Folge von Aktionen ohne weitere Eingaben durch den Benutzer. Eine reglementierte Transaktion ergibt sich als ein gerichteter Graph aus diesen Grundelementen mit einer Quelle (BOT) und mehreren Senken (EOT). In den Senken ist mindestens eine der Teilbedingungen aus der Nachbedin-

*) Brown (<Bro82>, S.415) führt als ein zusätzliches Konstrukt eine modifizierte Case-Anweisung ein, die sich auf eine in eine While-Schleife eingebettete Case-Anweisung zurückführen läßt.

gung der Spezifikation erfüllt.

Abb. 4-8.:

(* Kontrollstrukturen für den Dialog, vgl. <Bro82>, S.413 *)



(Zu den Regeln für die Produktion zulässiger Graphen aus den Grundelementen vergleiche Brown <Bro82>, S.414.)

Zur Implementierung von Transaktionen ist dieser Ansatz aus der Dialogprogrammierung um eine Komponente zu erweitern, die ein sinnvolles lokales Zurücksetzen der Transaktion ermöglicht. Im System R (<Gra77>, S.423) wurde hierzu die Idee eines "logischen" vom Benutzer angebbaren Stützpunkts "Savepoint" eingeführt. Die Transaktion wird im Fehlerfall oder bei Benutzer initiierten Abbrüchen der Ausführung nicht auf den Anfang, sondern auf diesen Stützpunkt zurückgesetzt.

Die Stellen im Transaktionsablauf, an denen Eingaben durch den Benutzer erfolgen, eignen sich zur Anwendung des Konzepts der lokalen Stützpunkte. Bei lose reglementierten Transaktionen kann es wünschenswert sein, in einem verfolgten Pfad, der sich nach Meinung des Benutzers als ungünstig erwiesen hat, die Transaktion nicht einfach abubrechen. Es ist oft günstiger, den Pfad in einer Folge geordneter, lokaler Rücksetzvorgänge aufzugeben und an einem früheren Zustand, der vom Benutzer ausgewählt wird, wieder aufzusetzen.

Die Definition von lokalen Stützpunkten und dieses partielle UNDO gilt es in der Implementierung zu berücksichtigen. Vom Datenverwaltungssystem ist dieses Vorgehen zu unterstützen.

Für die Implementationsstruktur einer reglementierten Transaktion ergibt sich eine Folge der im folgenden skizzierten elementaren Strukturen:

TRANSACTION transaction.

IMPLEMENTATION;

SCOPE;

:

BEGIN-OF-TRANSACTION

IF NOT <pre-condition> THEN UNDO END;

:

:

SAVEPOINT:

WHILE <condition>

DO CASE case (DC-Action(<userinput>) COND BACKUP) = A(x)

OF

A(1): <actions>;

:

```
        A(i):      <actions>;
                :
        A(n):      <actions>;
        ELSE:      <actions>;
    END;

SAVEPOINT:
IF <condition> AND
    bool(DC-Action(<userinput>) COND BACKUP)=TRUE;
    THEN <actions>;
    ELSE <actions>;
:
:
IF NOT <post-condition>
    THEN UNDO
    ELSE END-OF-TRANSACTION;
END transaction.
```

Der Gedanke der lokalen Stützpunkte wird durch das Sprachkonstrukt SAVEPOINT, der eine Marke für das Wiederaufsetzen darstellt, und durch den Zusatz COND BACKUP an einer DC-Aktion realisiert. Die bedingte Eingabeaufforderung (DC-Aktion) muß dabei in der durch den SAVEPOINT gekennzeichneten Anweisung enthalten sein. Durch den Zusatz COND BACKUP wird zum Ausdruck gebracht, daß der Benutzer an dieser Stelle durch seine Eingabe die Transaktion zum vorhergehenden Stützpunkt zurücksetzen kann. Die Effekte der nach diesem Stützpunkt ausgeführten Aktionen sind nach dem Rücksetzen nicht mehr existent.

Die Realisierung des Schreibens von Stützpunkten soll nicht näher ausgeführt werden. An dieser Stelle sei auf entsprechende Möglichkeiten, die durch die Recovery-Behandlung von Transaktionen bekannt sind, hingewiesen <Reu81>.

Ein Beispiel für die mögliche Implementierung einer reglementierten Transaktion findet sich im Anhang 3 als "Erstellen eines Tagesberichts".

Unter Umständen läßt sich durch reglementierte Transaktionen die Komplexität des Programms reduzieren. Durch die Eingabe der Werte von Bedingungsvariablen müssen komplexe Zusammenhänge und Auswertungen u.U. nicht programmiert werden.

Die Integration der reglementierten Transaktionen in das Datenbank-Anwendungssystem erfolgt völlig analog zu den schematischen Transaktionen (vgl. Kap. 6). Unterschiede ergeben sich in bezug auf die Überwachung der Integrität (vgl. Abs. 5.5).

Durch das Reglement einer Transaktion wird eine Menge möglicher Transaktionen festgelegt. Die Teilprädikate in der Nachbedingung der Transaktion und die eventuell lokalen Bedingungen für die Zwischenschritte sind für die Struktur entscheidend. Im Gegensatz dazu stehen die im folgenden zu betrachteten freien Transaktionen, deren transaktionsspezifische Nachbedingungen dem Datenbank-Anwendungssystem unbekannt sind.

Bemerkung: (Die Erzeugung von Programmen bezüglich der Gestaltung des Dialogs kann auf vielfältige Weise unterstützt werden. Eine Möglichkeit wäre zum Beispiel der Einsatz von Dialoggeneratoren, wie er von Kaiser/Stetina <KS82> vorgeschlagen wird.)

4.4. Freie Transaktionen

Bei schematischen und auch bei den reglementierten Transaktionen wurde davon ausgegangen, daß die beabsichtigten Effekte der Transaktion apriori bekannt sind. In den Datenbank-Anwendungen gibt es jedoch den häufigen Fall, daß für eine bestimmte Bearbeitung kein Algorithmus existiert und der Effekt der auszuführenden Transaktion bei der Initiierung noch unbestimmt ist. Erst am Ende einer Transaktion wird entschieden, ob die erzeugte Wirkung beabsichtigt war. Diese freien Transaktionen sind in ihrem Aufbau zum Zeitpunkt der Ausführung noch unbestimmt. Sie unterliegen im wesentlichen nur dem Willen des Benutzers und den innerhalb der Anwendung geltenden Normen (vgl. Abs. 2.1.3).

Zu dieser Art von Transaktion gehören insbesondere rein lesende Transaktionen, wie sie in der Form spontaner Abfragen durch den Benutzer gegeben sind, die keine Veränderungen auf der Datenbank vornehmen. Der Effekt solcher Transaktionen auf die Miniwelt ist jedoch nicht zu vernachlässigen (vgl. hierzu Aspekte des Datenschutzes <Schl81>).

Zur Realisierung der freien Transaktionen ist es notwendig, im Dialog auf die Aktionen der Datenbank-Anwendungsmoduln oder auf die elementaren Datenbank-Operationen zurückzugreifen. Dies gilt insbesondere für freie Auswertungen der Datenbank nach nicht vorhersehbaren Gesichtspunkten.

4.4.1. Rekonstruktion

Freie Transaktionen sind auf kreatives Handeln zurückzuführen. Es besteht prinzipiell die Möglichkeit neue Handlungen aus bekannten Teilhandlungen zu synthetisieren. Aber auch kreatives Handeln besitzt seine Regeln, eben die Einhaltung dieser Regeln kann von Datenbank-Anwendungssystemen noch unterstützt werden.

Eine neue Handlung wird konstituiert im Hinblick auf einen Zweck, wenn man zweckgerichtetes Handeln voraussetzt. Gelingt es die verfolgten generellen Zwecke einer Anwendung zu formalisieren, so kann eine ausgeführte neue Handlung daran gemessen werden, ob sie den Zwecken dienlich war. Aus Abs. 2.1 ist zu folgern, daß dies der Aussage entspricht, derzufolge neue Handlungen den Normen und Vorschriften der Anwendung genügen müssen. Die freien Transaktionen sind zu überwachen, ob sie diese Normen und Vorschriften einhalten.

Auch neue Handlungen werden durch Komposition aus bekannten Teilhandlungen erzeugt. Für freie Transaktionen ergibt sich daraus, daß sie entweder aus den elementaren Datenbank-Operationen zu bilden oder auf den Aktionen der Datenbank-Anwendungsmoduln aufzubauen sind.

4.4.2. Strukturelle Darstellung und Implementierung

Die freien Transaktionen entsprechen einem relationalen Übergang aus einer Menge möglicher Anfangszustände in eine Menge möglicher Endzustände der Datenbank. Die Möglichkeiten zur Strukturierung sind gering, da im Gegensatz zu den reglementierten Transaktionen die Muster der möglichen Übergänge ebenfalls unbekannt sind.

Freie Transaktionen können auf einer vorhandenen Sprachschicht der Datenbank kreiert werden. Dem Benutzer wird hierzu entweder eine operationale Sicht (realisiert durch die Datenbank-Anwendungsmoduln) oder eine statische Sicht (realisiert durch ein Externes Schema) zur Verfügung gestellt.

Die Entwicklung einer freien Transaktion entspricht dem freien Erstellen eines Programms, dessen Nachbedingung zum Zeitpunkt der Erstellung noch nicht bekannt ist.

Allerdings gelten die invarianten Bedingungen bezüglich des Zustands der Datenbank auch vor und nach Ausführung einer freien Transaktion. Für ein Datenbank-Anwendungssystem ist es deshalb notwendig, diese invarianten Bedingungen zu überwachen. Dies führt prinzipiell erst zur Notwendigkeit einer Integritätsüberwachung in den Datenbanksystemen, da wie bereits gezeigt wurde, die Implementationen schematischer und reglementierter Transaktionen verifizierbar sind.

Zur Spezifikation

Eine Spezifikation freier Transaktionen ist nur sehr eingeschränkt möglich. Es kann die (operationale oder statische) Sicht, auf der eine freie Transaktion gebildet wird, als äußerer Rahmen angegeben werden.

Die "Spezifikation" für eine freie Transaktion beschränkt sich somit auf den Vereinbarungsteil (SCOPE), wie er bei schematischen und reglementierten vorhanden ist.

```
TRANSACTION transaktion-scope;
SCOPE;
FROM <db-appl-module> IMPORT <actions>;
:
:
CONDITIONS;
  PRE-CONDITION: TRUE
  POST-CONDITION: TRUE
  (* Der operationsabhängige Teil der Vor- und
     Nachbedingung einer freien Transaktionen ist
     stets erfüllt *)
END transaktion-scope.
```

Die Spezifikation einer freien Transaktion umfaßt alle potentiell möglichen Transaktionen, die sich in der transitiven Hülle der importierten Aktionen erzeugen lassen. Beschränkt werden die möglichen Übergänge nur durch die invarianten Integritätsbedingungen der Datenbank. Sie sind analog zu den vorherigen Transaktionsarten mit der in der Spezifikation angegebenen Vor- und Nachbedingung konjunktiv zu verknüpfen.

Durch die Angabe dieser Umgebung für die Bildung freier Transaktionen wird dem Benutzer eine tiefere Sicht im Anwendungssystem (im obigen Fall die der Datenbank-Anwendungsmoduln) transparent gemacht.

Da zur Kontrolle der vollzogenen Übergänge mittels freier Transaktionen nur die invarianten Bedingungen herangezogen werden können, wird die Relevanz der in Abs. 5.3.3 erläuterten finalen deontischen Integrität evident. Sie legt die invarianten Bedingungen über die Zulässigkeit von Übergängen zwischen Zuständen der Datenbank fest.

Zur Implementierung

Die Implementierung einer freien Transaktion erfolgt auf der spezifizierten Umgebung interaktiv bei ihrer Aktualisierung. Zur Wahrung der Atomarität ist die freie Transaktion mit einer BOT zu beginnen und mit EOT abzuschließen. (Die BOT/EOT-Behandlung wird in vielen konkreten Datenbanksystem bis dato noch nicht unterstützt (vgl. <SB83>).)

Läßt man die Formulierung freier Transaktionen auf den statischen Sichten oder der Datenbank selbst zu, liegt der klassische Fall einer frei verwendbaren Datenmanipulationssprache vor (unter Berücksichtigung der BOT/EOT-Logik). Für das Verwaltungssystem bleibt, einen Mechanismus zur BOT- und EOT-Behandlung bereitzustellen, an den wiederum die Prüfroutinen für die Integritätsüberwachung geknüpft sind.

Die verbleibenden Möglichkeiten der Überwachung freier Transaktionen wird dennoch wichtig in bezug auf die Sicherung der Integrität. (vgl. Kap.4 und 5)

4.5. Zusammenfassung der Aspekte zur Schematisierung von Transaktionen

Für das weitere Vorgehen bezüglich der Integritätskontrolle ist die vorgenommene Einteilung der Transaktionen nach der Art ihrer Schematisierbarkeit von Bedeutung.

(1) Schematische Transaktionen

Sie stellen einen funktionalen Übergang dar.

Ausgehend vom einen Datenbankzustand verknüpft mit den Werten der Eingabeparameter ist der durch die Transaktion zu erzeugende neue Datenbankzustand apriori bestimmt.

(2) Reglementierte Transaktionen

Sie stellen einen relationalen Übergang dar.

Ausgehend vom einen Datenbankzustand ist eine Menge möglicher neuer Datenbankzustände bestimmt, die die Transaktion erzeugen kann. Welcher der spezifizierten Endzustände erreicht wird, ist abhängig von den Eingabeparameterwerten, die jedoch ihrerseits vom Zustand der Datenbank abhängig sind.

Eine reglementierte Transaktion besteht aus einer Kette funktionaler Übergänge. Die funktionalen Teilübergänge sind in der Regel komplexer als elementare Datenbankoperationen.

Bei den reglementierten Transaktionen lassen sich streng und lose reglementierte unterscheiden. Eine streng reglementierte Transaktion ist auf eine schematische Transaktion zurückführbar, wenn man alle eventuell relevanten Datenbankzustände und Alternativen in den Eingabeparameterwerten berücksichtigt. Lose reglementierte Transaktionen sind mit augenblicklichen Entscheidungen während des Ablaufs verbunden, die zu Beginn der Transaktion nicht angebbar sind.

(3) Freie Transaktionen

Sie stellen ebenfalls einen relationalen, allerdings nicht determinierten Übergang dar.

Die Menge der erzeugbaren Endzustände der Datenbank ist nur in einem losen Rahmen über die transitive Hülle der zur Verfügung stehenden Operationen und durch die Einhaltung der invarianten Bedingungen der Datenbank fixiert.

5. Integritätskontrolle in Datenbank-Anwendungssystemen

Der Begriff der Integrität der Datenbank war in seinem ursprünglichen Sinn auf die korrekte Abspeicherung der Daten und deren Erhalt bei eventuell auftretende Fehler in der Hard- und Software beschränkt. Der Integritätsbegriff wurde erweitert, um auch Fehler zu verhindern, die sich aus einer inkorrekten Benutzung der Datenbank ergeben. Die Integrität wurde deshalb in ihrer operationalen Seite behandelt als Realisierung der Isolation von Transaktionen (<Här78>). Im statischen Bereich wurde die Integritätsüberwachung auf die Wahrung der semantischen Integrität ausgedehnt (<Web81>), als eine Möglichkeit die zulässigen Zustände der Datenbank zu beschränken.

Im Rahmen dieses Kapitels wird versucht die Anforderungen an die Integrität für ein Datenbank-Anwendungssystem in seiner Gesamtheit zu betrachten. Eine erweiterte Betrachtung der Integrität aus handlungstheoretischer Sicht ermöglichte das zugrunde liegende Konzept, daß Handlungen durch Transaktionen repräsentiert werden. Dieses Vorgehen führte zu einer Ergänzung der Anforderungen an die Integritätskontrolle.

Im ersten Abschnitt wird deshalb eine Rekonstruktion der allgemeinen Integritätsanforderungen aus der Miniwelt heraus versucht. Die bekannten Integritätseigenschaften einer Datenbank werden explizit gemacht. Aus den handlungstheoretischen Überlegungen ergibt sich zusätzlich eine Erweiterung der Anforderung an die Integritätswahrung eines Datenbank-Anwendungssystems, die in dieser Form in den bekannten Ansätzen zur Integritätskontrolle noch keine Beachtung fand. Diese einzuführende Integritätsart basiert auf der Überwachung der für eine Anwendung geltenden Normen.

Die Struktur der einzelnen Integritätsarten und die Möglichkeiten ihrer Realisierung werden darauf aufbauend in den Abs. 5.2 - 5.5 von der Datenbank bis zum Transaktions-

system erörtert. Der Abschnitt 5.5 setzt die Integritätsüberwachung zu den verschiedenen Transaktionsarten in Beziehung. Es werden spezifische Lösungen und deren Grenzen bezüglich der einzelnen Transaktionsarten aufgezeigt.

5.1. Rekonstruktion der Integrität

Für den Aufbau einer Integritätskontrolle sind die Anforderungen an die Integritätswahrung kritisch zu rekonstruieren. Im einleitenden Abschnitt 1.3 wurden vier Möglichkeiten zur Klassifikation der Integrität in einem Datenbank-Anwendungssystem angegeben. An der Konstitution der Begriffe bei der Festlegung der Datenstrukturen für das Konzeptionelle Schema orientierte sich die Einteilung in konstitutive und regulative Integritätsanforderungen <Wed83a>. Eine analoge Vorgehensweise ergibt sich für die Konstitution von Handlungen beim Entwurf der Transaktionen. Die Unterscheidung in konstitutive und regulative Integrität erlaubt festzuhalten, welche Anforderungen für die Struktur der Daten oder Transaktionen bestimmend sind.

Änderungen in den konstitutiven Bedingungen erzeugen einen neuen Objekt- oder Handlungstyp. Änderungen regulativer Bedingungen modifizieren nur den Gebrauch der Begriffe und Handlungen ohne für sie substantiell zu sein (vgl. dazu <Wed83a>). Für den Aufbau einer Integritätsüberwachung sind Vorkehrungen zu treffen, die es erlauben regulative Integritätsanforderungen im System einfach änderbar zu gestalten.

Die Möglichkeiten zur Umsetzung der Integritätsanforderung in ein Kontrollsystem setzt daneben eine Klassifikation nach der Ebenenstruktur des Datenbank-Anwendungssystem voraus. Im weiteren werden zuerst die bekannten Arten der technischen, semantischen und strukturellen Integrität betrachtet. Anschließend wird eine auf der Handlungsbegründung

und -rechtfertigung (vgl. 2.1.3) basierende Integritätsart - die "deontische" Integrität - einführt.

5.1.1. Die semantische, strukturelle und technische Integrität aus der Sicht der Anwendung

Die im einführenden Abs. 1.3 skizzierten, aus der Literatur (vgl. hierzu <EC75>, <Sto75>, <Thu80>, <Dat81a>, <FSW81>, <Reu81>, <Web81>) bekannten Integritätsarten werden im folgenden in ihrer Beziehung zur Miniwelt näher untersucht. Es wird deutlich, daß sich die weit vorangetriebenen Überlegungen im Rahmen der Behandlung der Datenintegrität auch auf die Operationen übertragen lassen. Die Beziehung zwischen der Miniwelt und dem Datenbank-Anwendungssystem spielt für die Datenintegrität eine besondere Rolle. Die Motivation für integritätserhaltende Maßnahmen der operationalen Integrität wird insofern gegeben, als sie in Beziehung zu den handlungstheoretischen Grundlagen gebracht werden. Die Bezugnahme auf die Handlungstheorie erfolgt, insoweit sie für die Klassifikation und das Verständnis der Integritätskontrollmaßnahmen erforderlich erscheint.

Betrachtet man die einzelnen Integritätsarten im Hinblick auf die Daten und die Operationen über die einzelnen Ebenen eines Datenbanksystems hinweg, so erhält man die bekannten sechs Fälle.

(1.1) technische Datenintegrität

Die Gewährleistung der Korrektheit der Datenrepräsentation über die verschiedenen Ebenen eines Datenverwaltungssystems hinweg auch bei Störungen in den Hardwarekomponenten ist der klassische Fall der Sicherung der Datenintegrität. Für die Gewährleistung dieser Art der Datenintegrität existieren in den Datenbanksystemen ausreichende Mechanis-

men. Eine ausführliche Beschreibung gibt Reuter in <Reu81>.

Die Speicherung der Daten muß gegen Verfälschung oder Verlust durch technische Fehler, wie Plattenfehler, Stromausfall etc. unempfindlich sein. Es ist zu jedem Zeitpunkt zu gewährleisten, daß ein speicherkonsistenter Zustand der Datenbank existiert bzw. wieder regeneriert werden kann. Die technische Konsistenz umfaßt auch die korrekte Umsetzung der logischen Datenstrukturen (einschließlich der Zugriffspfade) in physische Datenstrukturen. Bei eventuellen technischen Fehlerfällen ist auch sie aufrechtzuhalten.

Im weiteren Verlauf der Arbeit wird diese Art der Integrität als gegeben vorausgesetzt. Die vielfältigen Probleme, die damit verbunden sind, und der doch beträchtliche Aufwand, der hierfür nötig ist, sind jedoch nicht gering zu schätzen.

Aus der Sicht der Anwendung bedeutet die Wahrung der technischen Datenintegrität, die Gewährleistung der Zuverlässigkeit des Mediums auf dem die Daten abgelegt sind.

(1.2) strukturelle Datenintegrität

Jedem Datenmodell liegt eine gewisse Struktur zugrunde, in der die Begriffe der Miniwelt darzustellen sind. Jedes Datenmodell stellt dabei Restriktionen an die Form der Darstellung.

Im Relationalen Datenmodell entspricht einer Relation jeweils ein Begriff (Objekttyp) aus der Anwendung (vgl. <Wed81>, S.98). Das Relationale Datenmodell erlaubt somit den zu einem Begriff gehörenden Objekten jeweils ein Tupel der diesen Objekttyp repräsentierenden Relation zuzuordnen. Daraus ergeben sich bestimmte Restriktionen an den Aufbau von Objekttypen (vgl. <Wed81>, S.172). Nachdem ein konzeptionelles Schema für eine Anwendung aufgebaut ist, sind im

Rahmen der strukturellen Datenintegrität zusätzliche Struktureigenschaften des Datenmodells abzusichern. Die einzelnen Tupeleinträge in den Relationen haben diese modellspezifischen Eigenschaften zu erfüllen.

Codd <Cod79> kennt vier Eigenschaften, deren Einhaltung für die Tupeleinträge im Relationalen Modell erfüllt sein müssen.

- Typintegrität

Jedem Attribut in einer Relation ist ein Wertebereich (Domain) zugeordnet. Die einzelnen Attributwerte in den Tupeln müssen aus den zugehörigen Wertebereichen sein.

- Schlüsselintegrität

In jeder Relation gibt es ein oder eine Kombination von Attributen, die als Schlüssel ausgewiesen sind, und damit jedes Tupel der Relation eindeutig identifizieren. Diese Schlüsseleigenschaft muß immer erfüllt sein. Beim Datenbankentwurf (vgl. <Wed81>) entspricht dies der Eindeutigkeit von Nominatoren.

- referentielle Integrität (nach Codd)

Die referentielle Integrität (nach Codd) ist bei Relationen mit einem einen zusammengesetzten Schlüssel (komplexe Relationen) zu beachten. Die komplexe Relation entsteht beim Datenbankentwurf (vgl. <Wed81>) aus der Komposition von Begriffen. Für die zur Komposition verwendeten Objekttypen können ebenfalls (Basis-)Relationenschemata im Konzeptionellen Schema existieren. Für die Tupeleinträge in der komplexen Relation haben in den zugehörigen Basisrelationen für die einzelnen Teile des komplexen Schlüssels korrespondierende Einträge zu existieren. Für jeden Tupeleintrag in der komplexen Relation gibt es somit auch einen entsprechenden Eintrag in den zugehörigen Basisrelationen.

- semi-referentielle Integrität

(Sie wird von Codd nur implizit gefordert.)

Der Fall der semi-referentiellen Integrität tritt auf, wenn zwei Relationen existieren, deren Schlüsselattribute

den gleichen Wertebereich besitzen und die eine (semi-abgeleitete) Relation bezüglich des Schlüssels eine Teilmenge der anderen (Basisrelation) ist. Die Relationen können sonst jedoch verschiedene Attribute besitzen. Für die semi-abgeleitete Relation ist zu fordern, daß nur Tupel-einträge zulässig sind, für die auch korrespondierende Einträge in der Basisrelation vorhanden sind.

Diese Integritätsforderung ergibt sich insbesondere beim Datenbankentwurf aus der Darstellung von Art-Gattungsbeziehungen. Die Objekte der Artbegriffe fallen ebenfalls unter den Gattungsbegriff. Die sonstigen Eigenschaften bezüglich der Art und der Gattung differieren jedoch (vgl. <Wed81>).

Die Codd'schen Forderungen sind um zwei Anforderungen zu erweitern, die sich ebenfalls auf Kriterien des Datenbankentwurfs beziehen. Sie berücksichtigen gängige Erweiterungen des Relationalen Datenmodells.

- **referentielle Integrität (nach Date)**

Eine Erweiterung der referentiellen Integrität nach Codd hat Date vorgenommen <Dat81a>. Es wird die Integrität der sogenannten Fremdschlüsselbeziehungen behandelt. Existiert zu einem Nichtschlüsselattribut einer Relation (Fremdschlüsselattribut) eine korrespondierende (Basis-)Relation, so hat für jeden Wert des Fremdschlüsselattributs ein gleicher Werteintrag im Schlüssel der Basisrelation zu existieren. Diese Beziehung ergibt sich beim Datenbankentwurf u.a. aus der Begriffsreduktion. Durch die Zugehörigkeit verschiedener Objekte eines Objekttyps zu einem Objekt eines anderen Objekttyps zum Ausdruck gebracht wird.

- **referentielle Integrität (nach Smith/Smith)**

Durch Smith/Smith <SS77> werden Begriffshierarchien in das Relationale Datenmodell eingeführt. Es wird möglich, Namen von Relationen als Attributwerte zu benutzen. Für die entsprechenden Attribute muß gelten, daß nur Namen von Relationen eingetragen werden, die im Konzeptionellen

Schema vorhanden sind. Auch diese Beziehung entsteht aus der Art-Gattungsbeziehung, indem die Eigenprädikatoren der Artbegriffe den Objekten der Gattung als Eigenschaft zugeordnet werden.

Alle dargestellten Integritätsbedingungen entsprechen Beziehungen, die durch das Relationale Datenmodell oder gängigen Erweiterungen (Date <Dat81a> und Smith/Smith <SS77>) unterstützt werden. Die Bedingungen fixieren die Struktur des Konzeptionellen Schemas und werden deshalb als strukturelle Integritätsbedingungen bezeichnet. In der gängigen Literatur (z.B. <FSW81>) werden sie allerdings von den folgenden semantischen Integritätsbedingungen nicht explizit unterschieden.

Die Bedingungen gewährleisten einen korrekten syntaktischen (bei Wertebereichsangaben) und grammatikalischen (bei den anderen Bedingungen) Aufbau der Daten im Sinne des zugrunde gelegten Datenmodells.

Implizit wird im Relationalen Datenmodell Redundanzfreiheit der Daten gefordert. Die referentielle Integrität garantiert im Grundprinzip bei nicht redundanten Einträgen von Werten deren Gleichheit. Wird die Redundanzfreiheit im Konzeptionellen Schema weitergehend verletzt, so ergeben sich weitere Datenreferenzen und damit auch weitere Arten "referentieller" Integritätsbedingungen.

(1.3) semantische Datenintegrität

Unter den Begriff der semantischen Datenintegrität wird in der Literatur mittlerweile eine ganze Reihe von Integritätsanforderungen subsumiert (z.B. auch die strukturelle Integrität (vgl. z.B. <Dat81>) aber auch einige Aspekte der operationalen Integrität <Web81>).

Die semantische Datenintegrität umfaßt in ihrem ursprünglichen Verständnis <EC75> die Beschränkung der im Konzeptionellen Schema fixierten möglichen Datenbankzustände. Die möglichen Datenbankzustände ergeben sich im Relationalen Modell aus dem Cartesischen Produkt der Wertebereiche der Attribute und aus den Einschränkungen der strukturellen Datenintegrität. Durch die semantische Integrität sind diese möglichen Zustände auf eine "sinnvolle" Teilmenge weiter einzuschränken. Die in der Datenbank abgelegten Daten sollen semantisch widerspruchsfrei sein. Anfragen an die Datenbank dürfen in einem Zustand nicht zu widersprüchlichen Aussagen für die Miniwelt führen.

Daneben soll durch die semantische Integrität erreicht werden, die Datenbankzustände auf jene Zustände einzuengen, deren dargestellte Sachverhalte in der Miniwelt auch wirklich auftreten können. (Die Bezeichnung "semantisch" deutet dies an.) Die Frage, ob eine dargestellte Situation fingiert, d.h. empirisch falsch ist, oder ob sie wirklich ist, ist jedoch durch die semantische Integrität nicht entscheidbar.

Selbst in der dargestellten Form bieten sich in einer Anwendung stets eine sehr große Vielfalt von Möglichkeiten zur Angabe semantischer Integritätsbedingungen. Semantische Integritätsbedingungen ergeben sich beim Datenbankentwurf vorallem aus den regulativen Bedingungen zur Begriffsbildung (vgl. <Wed83a>).

In welchem Umfang semantische Integritätsbedingungen in einem Datenbank-Anwendungssystem realisiert werden ist eine Entscheidung des Datenbankentwurfs. Sie wirkt sich entscheidend auf das Leistungsverhalten des Datenbank-Anwendungssystems aus.

(2.1) technische operationale Integrität

Analog zur Datenrepräsentation ist die Korrektheit der Implementation der Datenmanipulationsoperationen und deren korrekte Ausführung zu gewährleisten. Die Behandlung dieser Art von Integrität findet im Rahmen der Betrachtungen zur Fehlertoleranz und zur Zuverlässigkeit von Hard- und Softwaresystemen statt. So sind z.B. auftretende Hardwarefehler und sonstige Abbrüche durch geeignete Maßnahmen (Teile der Recovery-Maßnahmen) zu kompensieren. Im Rahmen dieser Arbeit wird die technische operationale Integrität als gegeben vorausgesetzt. Für das Datenbank-Anwendungssystem bedeutet dies auch, daß die BOT-/EOT-Behandlung im Datenverwaltungssystem korrekt ausgeführt wird.

(2.2) strukturelle operationale Integrität

Die strukturelle Integrität bezogen auf die Operationen in einem Datenbank-Anwendungssystem leitet sich aus den Prinzipien zur Transaktionsbildung ab. Die strukturelle operationale Integrität umfaßt die Einhaltung der Gray'schen Kriterien beim Aufbau und der Ausführung von Transaktionen.

Die Gray'schen Kriterien bestimmen die Übertragbarkeit von Handlungen aus der Miniwelt auf das Datenbank-Anwendungssystem (vgl. Abs. 2.1.4). Die sich ergebenden Beschränkungen sind für einen konsistenten Betrieb des Datenbank-Anwendungssystems notwendig.

Die einzelnen Kriterien bedingen folgende Anforderungen an die Integritätswahrung.

- Konsistenz (Isolation)

Aus der Forderung der Konsistenzerhaltung der Datenbank leitet sich (vgl. Abs. 1.2.2) die vollständige Isolation der einzelnen Transaktionen bezüglich der benutzten

Daten ab *). Es ist zu gewährleisten, daß die einzelnen Transaktionen während ihrer Ausführung völlig unabhängig von einander ablaufen. Es dürfen keinerlei Interdependenzen während des Ablaufs bestehen.

Diese Forderung bedeutet eine starke Restriktion bezüglich der als Transaktionen zu realisierenden Handlungen. Für die Anwendung ergibt sich, daß immer nur eine singuläre Handlung betrachtet wird. Die Existenz kooperierender oder konkurrierender Handlungen wird ausgeschlossen.

Für das Datenbank-Anwendungssystem ergibt sich als Konsequenz, daß keine kooperierenden Transaktionen als Einheiten formulierbar sind. Andererseits folgt aus der Isolation der Transaktionen eine einfache und handhabbare Überwachung der Zugriffe auf die Datenbank. Die Gewährleistung der strukturellen und der semantischen Datenintegrität wird dadurch vereinfacht **). Der Verzicht auf die Forderung nach Isolation der Transaktionen würde ihre Vernetzung bedeuten. Im Fehlerfalle würde dies aus der Forderung der Atomarität und der Dauerhaftigkeit bedeuten, daß alle beteiligten Transaktionen rückgängig zu machen bzw. zu kompensieren sind.

Das Kriterium der Isolation läßt sich durch geeignete Synchronisationsmechanismen (vgl. u.a. <Gra77>, <Hä78>,

*) Die Konsistenzforderung der Gray'schen Bedingung beschränkt sich operational auf die Isolation der Transaktionen. Die weiteren Eigenschaften der Konsistenzforderung wurden unter dem Aspekt der Datenintegrität bereits behandelt.

**) Die Forderung nach vollständiger Isolation der Transaktionen erscheint beim Einsatz von Datenbanken in der Prozeßdatenverarbeitung zu restriktiv. Durch die Isolation wird die Formulierung kooperierender Prozesse erschwert.

<GM82>) beim Zugriff der Transaktionen auf die Datenbank realisieren. Für die weiteren Überlegungen in der Arbeit wird die Gewährleistung der Isolation durch geeignete Synchronisationsmechanismen im Datenbanksystem vorausgesetzt und nicht weiter behandelt.

- Atomarität

Die Atomarität bildet das Hauptkriterium für die Übertragung von Handlungen in die Form von Transaktionen. Die Forderung nach Atomarität impliziert, daß eine Transaktion genau einer zu übertragenden Handlung der Miniwelt entspricht. Die Atomarität legt fest, welche Handlung durch welche Folge von Datenbank-Operationen als eine Einheit sowohl im Sinne der Miniwelt als auch der Datenbank zu behandeln ist. Die Gewährleistung der Atomarität wird durch die Einführung zusätzlicher Operationen, die als logische Klammer für eine Menge von ausgeführten Datenbank-Operationen wirken, durch das Datenverwaltungssystem realisiert. Die beiden klammernden Operationen, BOT (begin of transaction) und EOT (end of transaction) (vgl. <Gra77>, <Gra80>) sind durch die in Kap. 3 skizzierte Architektur eines Datenbank-Anwendungssystems entsprechend durchzureichen (vgl. auch Kap. 4).

Die Integritätsüberwachung hat für die Gewährleistung der Atomarität der Transaktionen Sorge zu tragen.

- Dauerhaftigkeit

In der Forderung nach Dauerhaftigkeit verbirgt sich, daß wenn eine Transaktion ordnungsgemäß abgeschlossen wird, das erzeugte Ergebnis in die Datenbank als verbindlich eingebracht wird. Die Dauerhaftigkeit einer Transaktion ist somit eng verknüpft mit der Verbindlichkeitserklärung für die Wirkung einer Transaktion.

In der Integritätsüberwachung ist zu gewährleisten, daß eine abgeschlossene Transaktion nicht mehr aus dem System entfernt werden kann. Das entscheidende Kriterium ist die Vertrauenswürdigkeit. Wurde eine Transaktion erfolgreich ausgeführt, kann der Benutzer darauf vertrauen, daß ihre Wirkung von Bestand ist.

Der erzeugte Zustand kann nur durch das Ausführen weiterer Transaktionen wieder verändert werden. Durch eine geeignete nachfolgende Transaktion kann insbesondere versucht werden, die Wirkung einer Transaktion zu kompensieren, falls ihr Ergebnis unerwünscht war.

Insgesamt soll die strukturelle operationale Integrität in Ahnlehung an Härder <Hä78> als Ablaufintegrität bezeichnet werden (wobei dort diese Bezeichnung im wesentlichen auf die Isolation beschränkt wird).

(2.3) semantische operationale Integrität

Die semantische Datenintegrität grenzt die möglichen Zustände in einer Anwendung aus, dem entspricht auf der operationalen Seite die Festlegung der möglichen Übergänge. Die Übergänge kann man dadurch spezifizieren, indem man angibt, unter welchen Voraussetzungen das Ausführen einer spezifischen Handlung (vgl. hierzu <Wri77>, S.86) zu welchem Resultat führt. Entsprechend ist für das Datenbank-Anwendungssystem anzugeben, unter welchen Voraussetzungen welche Wirkungen die einzelnen Transaktionen auf die Datenbank besitzen (vgl. Kap. 4). Es ist im Rahmen der Integritätsüberwachung sicherzustellen, daß der spezifizierte Übergang auch bei der Ausführung der Transaktionen eingehalten wird. Die Festlegung der zur Ausführung einer Transaktion notwendigen Bedingungen entspricht den Vorbedingungen der Spezifikation (vgl. Abs. 2.2 und Kap. 4). Die Wirkung wird durch die Nachbedingungen bestimmt. (Es ist noch einmal darauf hinzuweisen, daß bei einer verifizierten Implementa-

tion schematischer und reglementierter Transaktionen auf das Prüfen der Nachbedingung verzichtet werden kann.)

Die semantische operationale Integrität kann man somit als Übergangsintegrität auffassen. Sie entspricht den in der Programmierung üblichen Vor- und Nachbedingungen der Spezifikation. Auf spezielle Eigenschaften des Prüfens von Vor- und Nachbedingungen bei den reglementierten und freien Transaktionen wird noch eingegangen.

5.1.2. Eine Erweiterung der Integrität für Datenbank-Anwendungen

Häufig werden Integrität und Sicherheit als Zwillinge betrachtet. In dem Bereich zwischen Sicherheit und Integrität ist auch die im folgenden einzuführende Integrität anzusiedeln. Sie wird auf den in einer Anwendung geltenden Normen begründet.

Aus der Handlungsbegründung und Rechtfertigung ergibt sich, daß Normen das Handeln und damit letztlich auch die Zustandsübergänge einer Datenbank bestimmen. Normen sind Handlungsaufforderungen an alle Subjekte in bestimmten Situationen immer bestimmte Zwecke zu verfolgen (vgl. <Schw76>, S.28) *) Man kann sie auch als universelle generelle Aufforderungen bezeichnen.

Postuliert man, daß jedem Zustandsübergang der Datenbank ein gewisser Zweck bzw. Intention geordnet ist, (wie es für Handlungen in Abs. 2.1.3 deutlich wurde,) so ist auch innerhalb eines Datenbank-Anwendungssystem die Überwachung

*) Die Normen entstehen durch Genese der subjektbezogenen Maximen durch Verständigung auf gemeinsam zu verfolgende Zwecke, vgl. <Sch76>. Der Subjektbezug wird in Kap. 6 behandelt.

der Normen möglich.

Es gibt zwei mögliche Arten, um Normen zu postulieren:

- (1) finale Normen,
sie werden dargestellt durch die Aufforderung einen gewissen Zustand oder Situation zu erzeugen,
- (2) afinale Normen,
sie werden dargestellt durch die Aufforderung zur Aktualisierung von Handlungen.

Die beiden Arten von Normen finden sich in der Entwicklung verschiedener Normenlogiken (oder deontischen Logiken) wieder. In Analogie zur Darstellung der Normen unterscheidet v.Wright (<Wri77>, S.106) Logiken des Sein-Sollen-Typs und Logiken des Tun-Sollen-Typs.

Übertragen auf das Datenbank-Anwendungssystem bedeutet dies, daß sich zusätzlich zu den vorherigen im Abschnitt aufgeführten Integritätsarten eine weitere Kontrollmöglichkeit ergibt. Überwacht werden kann, welche Transaktionen ausgeführt werden dürfen oder ausgeführt werden sollen, um in einem konkreten Zustand einen den Normen gerechten Datenbankzustand zu erreichen.

In einem ersten Unterabschnitt wird auf die Möglichkeit der Darstellung und den Aufbau von Normen (vgl. <Schw76>) eingegangen, im zweiten wird auf die Möglichkeit einer Integritätskontrolle auf der Basis der deontischen Logik skizziert.

In diesem Zusammenhang ist auch auf das Datenschutzgesetz (<BD SG80>, S.27) hinzuweisen. Es enthält sowohl personenbezogene als auch personenfreie Anweisungen zur Überwachung der Verarbeitung (d.h. der Transaktionen) in einer Datenbank-Anwendung.

5.1.2.1. Finale und afinale Normen - der intentionale Aspekt

Analysiert man eine Anwendung, so ist ein Bestandteil des Handelns die Befolgung gewisser Anweisungen, um in bestimmten Situationen vorgegebene Zwecksetzungen zu erreichen. In der Handlungstheorie <Schw76> werden solche Anweisungen, wenn die Zwecke generell gelten und die Aufforderung zur Befolgung an alle Subjekte gerichtet ist, als Normen bezeichnet. Für die Überwachung von Normen in einem Datenbank-Anwendungssystem sind sie in einer geeigneten Form darzustellen. Wie bereits erwähnt gibt es prinzipiell zwei Möglichkeiten zur Festlegung der Normen - einen finalen und einen afinalen Ansatz.

Der finale Ansatz basiert auf der Formulierung der in der Anwendung verfolgten Zwecke in der Form von Zuständen. Innerhalb der Datenbank-Anwendung sind dies die erzeugbaren Zustände der Datenbank. Nicht erfaßt werden die Zustände der Objekte der Miniwelt außerhalb der Datenbank, obwohl sie eigentlich in die Überwachung mit einzubeziehen sind. Die finalen Normen stellen invariante Bedingungen dar, die für alle Zustandsübergänge erfüllt sein müssen.

Der afinale Ansatz basiert auf der Vorgabe von Handlungsschemata. Die Aktualisierung dieser vorgegebenen Handlungsschemata dient in bestimmten Situationen immer dem Erreichen der in den Normen fixierten Zwecke. Diese Festlegung von Normen auf afinale Weise setzt allerdings die Kenntnis der Handlungsschemata und -muster und damit auch deren Wirkung voraus. Für das Datenbank-Anwendungssystem wird somit die Darstellung der Handlung als schematische oder reglementierte Transaktionen erforderlich.

Aus der Festlegung der Normen ergibt auch für das Überwachungssystem ein finaler oder zustandsorientierter und ein afinaler oder operationaler Ansatz.

Eine konkrete Ausführung einer Transaktion ist an den vereinbarten Normen zu prüfen. Im Sinne der Handlungstheorie ist sie zu begründen und zu bewerten. Dies bedeutet, daß die Wirkung der Transaktion nicht nur die transaktionsspezifischen Nachbedingungen erfüllt, sondern mit den Normen der Anwendung verträglich ist. Deontische Integrität heißt, die Ausführung von Transaktionen befinden sich mit den geltenden Normen in Übereinstimmung.

Die Normen einer Anwendung besitzen in der Regel eine gewisse Struktur (vgl. <Schw76>) in ihren Beziehung untereinander. Es ist sinnvoll eine Hierarchisierung der Normen vorzunehmen und damit die Prüfbarkeit zu vereinfachen. Die finalen Normen erfordern nach einem durchgeführten Zustandsübergang das Prüfen der entsprechenden invarianten Bedingungen. Die afinalen Normen sind größtenteils bereits bei der Aktualisierung der Handlung (und damit der Transaktion) prüfbar. Dies ist ein entscheidender Vorteil.

(Es ist insbesondere die Verträglichkeit der einzelnen Normen zu fördern. Falls die Normen einer Miniwelt unverträglich sind, liegt eine (<Schw76>) Konfliktsituation vor. Für das Datenbank-Anwendungssystem ist die Normenstruktur so aufzubauen, daß das Auftreten von Konfliktsituationen ausgeschlossen wird (vgl. <Wed80>).)

5.1.2.2. Das Handeln nach Normen - der deontische Aspekt

Im Rahmen einer umfassenden Überwachung der Integrität innerhalb einer Datenbank-Anwendung sind die Möglichkeiten zur Überprüfung auf Einhaltung gültiger Normen der Miniwelt von Interesse. Die Überwachung bedarf einer Bewertung der Handlungen bzw. Transaktionen. Im konkreten Fall ist zu entscheiden, ob eine Handlung normenkonform ist.

Die Kategorien zur Bewertung von Übergängen auf Einhaltung der Normen sind nicht die logischen Modi "wahr" oder "falsch", sondern die modal-logischen Modi "geboten", "verboden" und "erlaubt". Man nennt diese auch die deontische Modi und die darauf aufbauenden Logiken "Deontische Logiken" oder "Normenlogiken" (vgl. <Wri77>). Es gibt eine Vielzahl von Ansätzen zum Aufbau deontischer Logiken (vgl. <Kal72>), denen zwei Prinzipien der Normendarstellung zugrunde liegen. Wie bereits erwähnt, führen afinale Normen zu einer Logik des "Tun-Sollens", in der über die Handlungen geurteilt wird. Die Darstellung finaler Normen ergibt dagegen eine Logik des "Sein-Sollens", in der Zustände beurteilt werden.

Für die Integritätsüberwachung ergeben sich somit auch zwei Arten der Kontrolle:

- finale deontische Integrität

Sie ist zustandsorientiert.

Es werden die erzeugten Datenbankzustände hinsichtlich der geltenden finalen Normen beurteilt und überwacht.

- afinale deontische Integrität

Sie ist operations- bzw. transaktionsorientiert.

Es werden die Aktualisierungen der Transaktionen bezüglich der geltenden afinalen Normen geprüft.

Eine Handlung (und damit eine Transaktion) oder ein Zustand in der Miniwelt (und somit auch in der Datenbank) ist entweder erlaubt, geboten oder verboten. Für das weitere Vorgehen ist es notwendig, Kriterien und Mechanismen zu fin-

den, die es ermöglichen konkrete Transaktionen und erzeugte Datenbankzustände bezüglich der deontischen Modi zu beurteilen. Abhängig von der Beurteilung ist dann entsprechend zu agieren bzw. zu reagieren.

Hinsichtlich der deontischen Modalitäten ergibt sich noch ein zusätzlicher Aspekt. Man kann wie Wright (<Wri77>, S.129) und andere <Kal72> zwischen einem offenen und geschlossenen (Rechts-)System unterscheiden. In einem geschlossenen System ist alles verboten, was nicht explizit erlaubt ist. In einem offenen System ist alles erlaubt, was nicht explizit verboten ist. Analog zu Wright (<Wri77>, S.129) ergibt sich dann eine starke Erlaubnis (explizit erlaubt) und eine schwache Erlaubnis (nicht verboten, jedoch nicht explizit erlaubt).

Für einen operationalen Aufbau der Integritätsüberwachung ist eine deontische Logik des "Tun-Sollen"-Typs heranzuziehen. An dieser Stelle wird auf der Normenlogik von Tammelo/Klinger (nach <Kal72>, S.121-126) aufgebaut, die mit der deontischen Logik Wrights <Wri77> verwandt ist. Tammelo unterscheidet, wie auch in Abs. 2.1.1 geschehen, zwischen der Aktualisierung und der Unterlassung einer Handlung. Tammelo teilt die Handlungen bei einem offenen System in vier Klassen ein, die den deontischen Modi entsprechen. (Für die Transaktionen gilt dies entsprechend.)

Die vier Klassen sind:

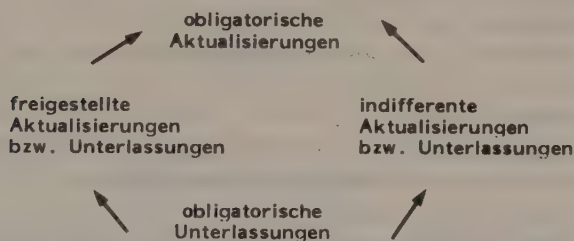
- (1) obligatorische oder gebotene Aktualisierungen
(identisch mit verbotenen Unterlassungen)
- (2) freigestellte oder erlaubte Aktualisierungen
bzw. Unterlassungen
(dies entspricht der starken Erlaubnis nach Wright)
- (3) indifferente Aktualisierungen bzw. Unterlassungen
(die der schwachen Erlaubnis nach Wright entsprechen)
- (4) obligatorische oder gebotene Unterlassungen
(identisch mit verbotenen Aktualisierungen)

Die drei Klassen (1), (2) und (4) bestimmen dabei ein geschlossenes System.

Abb. 5-1.:

(* deontische Handlungsklassen

nach Tammelo/Klinger <Kal72>, S.123 *)



Die vier Klassen kann man in Anlehnung an Tammelo wie folgt zusammenfassen:

- (1) und (2) zu befugten (autorisierten) Handlungen,
- (2) und (4) zu befugten (autorisierten) Unterlassungen,
- und
- (1), (2) und (3) zu zulässigen Handlungen,
- (2), (3) und (4) zu zulässigen Unterlassungen.

In einem konkreten Zustand der Datenbank ist zu beurteilen, unter welche Klasse eine beabsichtigte Transaktion, anhand der geltenden Normen fällt. *) Entsprechend

*) Der deontische Aspekt ist vom Aspekt der modalen Betrachtung der Möglichkeit zur Ausführung einer Transaktion klar zu trennen. Die Möglichkeit zur Ausführung beruht auf einer Modalisierung der semantischen Integrität.

dieser Einordnung ist vom Kontrollsystem zu reagieren. Es ist zu prüfen, welche Normen die Aktualisierung betreffen, und in welchem Modus sie zur betroffenen Norm steht. Dieser Vorgang wird explizit noch in Abs. 5.5 behandelt. Durch die afinale deontische Integrität wird bei einer anstehenden Aktualisierung folglich festgelegt,

ob sie erlaubt ist,

 sie kann damit ausgeführt werden;

ob sie verboten ist,

 die Ausführung ist durch das Kontrollsystem zu verhindern,

ob sie geboten ist,

 das Kontrollsystem hat dafür Sorge zu tragen, daß eine Aktualisierung der Transaktion stattfindet.

(Zum Anstoß einer Transaktion ist die Einführung von initialen Bedingungen erforderlich, die auf den Trigger-Mechanismus <Esw76> hinführen. Die weitere Behandlung erfolgt in Abs. 5.5.)

Die Aktualisierungen von Transaktionen, über die nicht aufgrund der geltenden Normen entschieden werden kann, sind als indifferent einzuordnen.

Geht man von einer operationalen Betrachtung zu einer zustandsorientierten Betrachtung über, erhält man eine entsprechende Einteilung der Zulässigkeit zur Erzeugung von Datenbankzuständen.

Die finale deontische Integrität ist eine Form der Integrität, die den ursprünglichen Datenbankzustand mit dem durch eine Transaktion erzeugten Zustand vergleicht. Sie bewertet den erzeugten Datenbankzustand hinsichtlich der durch finale Normen festgelegten Bedingungen.

Bei dieser zustandsorientierten Betrachtung ergibt sich völlig analog zur vorherigen Einteilung der Transaktionen die Unterscheidung der Datenbankzustände in gebotene, er-

laubte, verbotene und indifferente. Die Beurteilung ist jedoch abhängig, in welchen Ausgangszustand sich die Datenbank vor ihrer Erzeugung befand.

Zusammenfassend gilt, die Integritätskontrolle auf der Basis einer Normenlogik des Sein-Sollen-Typs erscheint anwendbar sinnvollerweise dann, wenn die Handlungsmuster unbekannt sind. Dies ist der Fall, wenn im Anwendungssystem freie (teilweise auch der reglementierten) Transaktionen zu überwachen sind. Eine Überwachung auf der Basis einer Normenlogik des Tun-Sollen-Typs erscheint sinnvoll dann, wenn die Transaktionen in schematischer Form und meist auch in reglementierter Form vorliegen. Bei den Prüfungen der drei Transaktionsarten wird in Abs. 5.5 entsprechend verfahren.

Für alle Betrachtungen galt das Universalitätsprinzip, es wurde kein Bezug zu Personen hergestellt, die die Handlungen oder Transaktionen ausführen oder initiieren. Die deontische Betrachtung der Integrität kann man diesbezüglich erweitern. Es können zur Beurteilung nicht mehr Normen sondern Vorschriften (Maximen) herangezogen werden, die subjektbezogen sind und nicht mehr universell gelten (vgl. Kap. 6).

5.2. Einige Prinzipien der Integritätskontrolle

Das Ziel des Aufbaus einer transaktionsorientierten Integritätskontrolle läßt sich nur erreichen, indem die im Abs 5.1 vorgenommene Klassifikation der Datenbank-Integrität detaillierter im Hinblick auf ihre Realisierbarkeit innerhalb eines Datenbank-Anwendungssystems betrachtet wird. Hierzu werden in diesem Abschnitt die prinzipiellen Voraussetzungen zur Integritätskontrolle diskutiert. Die Bedeutung der Bereitstellung von Mechanismen für die einzelnen Schichten des Datenbank-Anwendungssystem (Datenbank, statische Sicht, operationale Sicht, Transaktionen) wird aufgezeigt.

Im weiteren dieser Arbeit wird für die Betrachtungen zur Datenintegrität stets auf der Ebene der Datenbank aufgesetzt. Die gemachten Überlegungen zur Integrität beschränken sich somit auf die logische Ebene des Datenmodells und der Beziehung der Datenbank zur Miniwelt. Die Gewährleistung der (implementierungs-)technischen oder physischen Integrität (wie z.B. die korrekte Speicherung) wird nicht näher erläutert. Ausgiebige Untersuchungen hierzu finden sich in <Reu81>.

Die Kontrolle der Integrität wird bestimmt durch das Erkennen, daß Integritätsbedingungen verletzt sind und der anschließenden Reaktion darauf. Es stellt sich die Frage, wie in einem Datenbank-Anwendungssystem die Verletzungen erkennen lassen und auf welche Weise die notwendigen (Re-)Aktionen zur Beseitigung der Verletzung eingebracht werden können.

Die Verletzungen der Integrität sind hinsichtlich ihrer Wirkung auf Datenbank und Miniwelt zu unterscheiden. Für die Integrität der Datenbank gilt, daß nur das Ausführen (bei finaler deontischer Integrität auch das Unterlassen) von Änderungsoperationen die Integrität verletzen kann. Die Inte-

gritätskontrolle ist deshalb in erster Linie eine Kontrolle der Änderungsoperationen. Für die Miniwelt können allerdings auch Leseoperationen auf der Datenbank integritätsverletzend sein, wenn auf inkorrekte Daten Bezug genommen wird ("inkonsistente Analyse" nach <Hä78>).

Für die Integritätskontrolle gilt folgende prinzipielle Vorgehensweise:

- (1) Erkennen der Verletzung (Integritätsprüfung),
- (2) Erzeugen eines korrekten Zustands.

Eine mögliche Integritätsverletzung einer Operation kann vor Ausführung der Operation erkannt werden, wenn die Wirkung der Operation im voraus bekannt ist. Die Möglichkeit wird als "Apriori"-Erkennung bezeichnet. Ist die Wirkung der Operation apriori unbekannt, ist eine Prüfung der Integrität erst nach ihrer Ausführung möglich - "Aposteriori"-Prüfung. Die Möglichkeiten zur Korrektur sind wiederum abhängig von der Art der Erkennung. Im Falle der Apriori-Erkennung kann eine mögliche Verletzung bereits verhindert werden. Bei Aposteriori-Prüfung dagegen ist eine Korrektur notwendig. Sie ist entweder durch "Rücksetzen" der Operation (Herstellen des ursprünglichen Zustands vor Ausführung der Operation) oder durch Komplementieren der Operation (Ergänzen der ausgeführten Operation um weitere Aktionen, die einen integren Zustand erzeugen) zu realisieren. Die erstere Methode wird auch als "Backward-Recovery", die zweite als "Forward-Recovery" bezeichnet.

Erkennung der Integritätsverletzung

Zur Erkennung einer Integritätsverletzung ist der "Wirkungsbereich" einer Operation (bzw. Transaktion) einzugrenzen, den sie bei ihrer Ausführung "beeinflusst". Es ist die Kontrollsphäre der Operation bezüglich der Datenbank zu bestimmen. Es ist zu untersuchen, welche Zuordnung von Objekten der Datenbank zu den Operationen existiert. Für die Kontrolle sind die Objekte aus dem Argument- und Wertebereich der Operation relevant. Hierzu gilt es alle zugeordneten Integritätsbedingungen zu finden. Jeder Integritätsbedingung ist wiederum eine bool'sche Funktion zugeordnet (vgl. Abs. 1.1.1). Der Argumentbereich dieser Funktionen ist das Kriterium für die Zuordnung zur Kontrollsphäre einer Datenbank-Operation (oder dem "Betroffensein"). Die "betroffenen" Bedingungen ergeben sich über die Schnittmenge des Argument- und Wertebereichs einer Operation (vgl. Abs. 1.2.2) mit den Argumentbereichen der Integritätsbedingungen. Alle Objekte der Datenbank, die sich in Argumentbereich einer betroffenen Integritätsbedingung befinden, gehören zur Integritätskontrollsphäre der Operation (vgl. Abs. 1.3).

Die Integritätskontrollsphäre einer Transaktion umfaßt somit alle Objekte der Datenbank, auf die sie direkt

(durch die Ausführung elementarer Datenbank-Operationen)
oder indirekt

(durch Prüf- oder Folgeoperationen, die zur Überwachung
und zur Integritätssicherung der Transaktion nötig sind)
zugreift.

Im Kap. 1 wurden die Integritätsbedingungen formal sehr grob gefaßt, indem man als Argumentbereich der zugehörigen bool'schen Funktionen die gesamte Datenbank angab. Wie aus Abs. 5.1 zu erkennen war, bezieht sich eine Integritätsbedingung auf eine Menge von Relationen. Die Relationen werden miteinander direkt oder nach Anwendung eingebetteter Funk-

tionen in Beziehung gesetzt.

Die bool'sche Funktion zu einer Integritätsbedingung kann wie folgt geschrieben werden.

Sei $INT = \{ int-i \mid i \in N \}$ die Menge der Integritätsbedingungen einer Datenbank DB mit dem Definitionsbereich
 $DATABASE = (RELATION_1, \dots, RELATION_n)$,
dann gilt für alle $int-i$

$int-i : VIEW_i \rightarrow \{ TRUE, FALSE \}$
wobei eine Ableitungsvorschrift $rep-i$ existiert, so daß

$rep-i : RELATION_1 \times \dots \times RELATION_n \rightarrow VIEW_i$
mit $RELATION_i$ ist aus $DATABASE$.

Eine Datenbank-Operation $op-j$ läßt sich bezüglich der Datenbank schreiben, als eine Funktion *)

$op-j : RELATION_1 \times \dots \times RELATION_n \rightarrow RELATION_j$

Die von einer Datenbank-Operation betroffenen Integritätsbedingungen sind alle Bedingungen, deren Relationen aus dem Definitionsbereich der zugehörigen Repräsentationsfunktion mit den Relationen aus dem Definitionsbereich oder Wertebereich der Datenbank-Operation gemeinsame Elemente besitzen.

Die betroffenen Integritätsbedingungen einer Transaktion ergeben sich aus der Vereinigung der betroffenen Bedingungen ihrer Datenbank-Operationen. Die Kontrollsphäre einer konkreten Ausführung einer Transaktion umfaßt somit alle Objekte, die im Argumentbereich der betroffenen Integritätsbedingungen (indirekte Sphäre) oder im Argument- oder Wertebereich

*) Die dargestellte Form gilt für Änderungsoperationen. Bei Leseoperationen gibt es keinen Effekt bezüglich der Datenbank, für die Ergebnisrelation ist deshalb NIL zu setzen.

bereich der Transaktion selbst (direkte Sphäre) enthalten sind.

Die Kontrollsphären können weiter klassifiziert werden nach Art der Betroffenheit. Der Mechanismus zur Integritäts-sicherung ist davon abhängig, ob auf ein Datenbankobjekt nur lesend oder auch schreibend zugegriffen wird, leitet sich daraus ab, ob die Basisrelationen der Integritätsbedingung nur als Argumente in einer Datenbank-Operation oder auch in deren Wertebereich auftreten.

Die "Lesekontrollsphäre" einer Transaktion wird festgelegt durch alle Objekte, die als Argumente einer Datenbank-Operation auftreten. Die "Schreibkontrollsphäre" einer Transaktion ist bestimmt durch alle Objekte im Wertebereich von Operationen der Transaktion.

Abb. 5-2.:

(* Klassifikation der Integritätssphären *)

Art des Zugriffs			
		schreibend	lesend
Trans- aktion	Integritäts- bedingung	direkte Schreibsphäre	direkte Lesesphäre
		indirekte Schreibsphäre	indirekte Lesesphäre

Wie bereits erwähnt, führt die Integritätskontrolle ihrerseits im Rahmen einer Transaktion (ergänzende) Aktionen aus. Die durch diese ergänzenden Prüfoperationen berührten Objekte der Datenbank sind ebenfalls in die Integritätskontrollsphäre mit einbezogen.

Es ist offensichtlich, daß auch in der Miniwelt "Integritätsverletzungen" auftreten können. Diese entziehen sich jedoch bis auf die Kontrolle der Eingabe- und Ausgabeparameter der Überwachung durch das Datenbank- bzw. Anwendungssystem.

Es wurde aufgezeigt, wodurch festgelegt wird, welche Integritätsbedingungen durch eine Transaktion betroffen sind und welche Objekte der Datenbank damit in die Integritätsprüfung einer Transaktion einzubeziehen sind.

Ein weiterer Gesichtspunkt der Integritätskontrolle ist der Prüfzeitpunkt, wann eine Integritätsbedingung sinnvoll prüfbar ist. Da die Einheit zur Transformation von Datenbankzuständen die Transaktion bildet, sind die invarianten datenbankbezogenen Bedingungen zu Beginn und am Ende einer Transaktion gültig. Die Notwendigkeit, eine Prüfung der invarianten Bedingungen zu aktivieren, hängt von der Art der Transaktion ab, wie im Abs. 5.5 noch ausgeführt wird. So lassen sich frühestens alle Integritätsbedingungen bei schematischen zu Beginn, bei reglementierten während, und bei freien Transaktionen nach der Transaktionsausführung prüfen.

Alle von einer Transaktion betroffenen Integritätsbedingungen sind jedoch spätestens zum Zeitpunkt des EOT zu prüfen.

Korrektur von Integritätsverletzungen

Wird eine Integritätsverletzung erkannt, so kann stets auf einen integeren Datenbankzustand zurückgesetzt werden. Das Zurücksetzen einer Transaktion kann lokal, d.h. bis zum nächsten Stützpunkt (vgl. Abs. 4.3.2), oder total auf den Ausgangszustand vorgenommen werden. Es kann aber auch versucht werden, aus dem gegebenen Zustand durch ergänzende Aktionen (ohne Rücksetzen) einen integeren Datenbankzustand zu erzeugen.

Das Ergänzen von Transaktionen oder (auch von Aktionen, vgl. Abs. 5.4) durch weitere Aktionen erfordert das Einbringen sog. Regeln <SJR81> oder Trigger <Esw76> in das Datenbanksystem. Es wird aufgrund eines eingetretenen Ereignisses (z.B. eines bestimmten Zustands der Datenbank) eine Aktion aktiviert. Zur Formulierung von Triggern genügt es, die Integritätsbedingungen zu negieren und um eine Aktion zu erweitern. Bei Erfüllung dieser Bedingung wird die Aktion ausgeführt.

Der Triggermechanismus kann auch verwendet werden, um ganze Transaktionen anzustoßen. Dies ist z.B. im Rahmen der deontischen Integrität zur Aktivierung gebotener Transaktionen erforderlich.

Die Notation der Triggermechanismen erfolgt in Analogie zu <SJR81>:

```
TRIGGER triggername  
  ON <condition>  
  THEN <action>.
```

Die Einbindung dieser integritätssichernden Regeln (Trigger) wird in den folgenden Abschnitten jeweils spezifisch zu den einzelnen Schichten des Datenbank-Anwendungssystems behandelt.

Notwendigkeit einer schichtenspezifischen Integritätskontrolle

Die im einführenden Abs. 1.3 und im Abs. 5.1 skizzierten Integritätsarten sind, wenn man sie im Hinblick auf ihre Kontrollierbarkeit betrachtet, den einzelnen Ebenen des Datenbank-Anwendungssystems zuzuordnen. Im Kap. 3 wurde die Architektur eines Datenbank-Anwendungssystems entwickelt, die sich von der Datenbank ausgehend über die Sichten auf die Datenbank, den anwendungsorientierten Datenbank-Moduln bis zu den Transaktionen erstreckt. Für jede dieser einzelnen Ebenen sind spezifische Mechanismen zur Erkennung von Integritätsverletzungen und insbesondere Mechanismen zur Integritätssicherung bereitzustellen.

Die Abbildung 5-3 illustriert auf den welchen Ebenen des Datenbank-Anwendungssystems welche Bedingungen formulierbar sind. Es wird die auf jeder Ebene die mögliche operationale, Prüfbarkeit illustriert, die bei Ausführung einer Operation gegeben ist.

Die statischen oder objektorientierten Aspekte der Integrität werden auf der Ebene der Datenbank und der Sichten formuliert (Abs. 5.3). Die operationale Seite kommt mit der Einführung der operationalen Sicht der Datenbank hinzu (Abs. 5.4) und findet ihren Abschluß in der Betrachtung der Möglichkeiten zur Integritätskontrolle bezüglich der einzelnen Transaktionsarten (Abs. 5.5).

Abb. 5-3.:

(* Typische Bedingungen für die Ebenen des Anwendungssystems und deren operationale Prüfbarkeit *)

Ebene im Anwendungssystem	Formulierbarkeit	operationale Prüfbarkeit
Datenbank	invariante Bedingungen	sehr gering, nur elementare DB-Operationen
Anwendungsmoduln	Aktionsbedingungen, lokale Invarianten	teilweise, auf Aktionen
Transaktionssystem	Übergangsbedingungen, deontische Bedingungen	alle, da Transaktionen

Sowohl die strukturelle, die semantische Datenintegrität *) und die finale deontische Integrität sind auf der Basis der Datenbank und der Sichten zu formulieren. In Beziehung zu den Transaktionen sind sie über den Zeitpunkt ihrer Gültigkeit zu setzen. Sie sind zu untersuchen, ob sie temporär nur nach der Ausführung von Datenbank-Operationen, Aktionen oder Transaktionen erfüllt sind. Die Zeitpunkte, an denen eine Überprüfung stattfinden muß bzw. sinnvoll erscheint, ist somit operationsspezifisch bestimmt (vgl. Abb. 5-3).

*) In dieser Arbeit wird auf die Betrachtungen zur Sicherung der Integrität auf den tieferen Ebenen im Datenverwaltungssystem (hier als technische Integrität bezeichnet) bewußt verzichtet.

Die operationale Integrität ist erst bei der Betrachtung der Aktionen von Interesse. Die Forderung der Korrektheit des Übergangs (Einhaltung der Vor- und Nachbedingungen) gilt für die Aktionen der Datenbank-Anwendungsmoduln ebenso wie für die Transaktionen. Die Forderung nach Einhaltung der strukturellen operationalen Eigenschaften ist für die Aktionen nur lokal zu sehen. Sie wird bei der Transaktion global für das gesamte Anwendungssystem im integritätswahrenden Sinne. Die afinale deontische Integrität bezieht sich auf das Transaktionsniveau und kann im Rahmen der Betrachtung von Aktionen (insbesondere bei reglementierten Transaktion) verfeinert werden.

5.3. Integritätskontrolle auf der Datenbank und in den Sichten

Im Mittelpunkt dieses Abschnitts steht die Behandlung der verschiedenen Gesichtspunkte der Datenintegrität auf der Ebene der Datenbank und der Datenbanksichten. Im Rahmen dieser Arbeit ist es allerdings nicht möglich alle Integritätsaspekte mit der gleichen Ausführlichkeit zu behandeln. Integritätsarten, über die bereits mehrere Arbeiten existieren, werden der Vollständigkeit wegen nur andiskutiert. Die Aspekte der finalen deontischen Integrität werden dagegen besonders hervorgehoben.

Die Betrachtungen zur Datenintegrität setzen auf der Ebene des Konzeptionellen Schemas der Datenbank auf und reichen bis zur Beziehung des Anwendungssystems zur Miniwelt. Die verschiedenen Integritätsarten und entsprechenden Mechanismen zu ihrer Kontrolle werden im folgenden auf ihre operationale Realisierbarkeit untersucht.

Neben der Behandlung der einzelnen Integritätsarten ist die Integritätskontrolle bezüglich des Externen Schemas und der damit verbundenen "lokalen" Datenbank relevant. Die Beziehung einer "lokalen" Integrität bezüglich des Externen Schemas und den "globalen" Integritätsanforderungen bezüglich der Datenbank als Ganzes wird in Abs. 5.3.4 behandelt. Dies ist auch im Hinblick auf eine konsequente Anwendung der ANSI-SPARC-Architektur <Yor76> von Datenbanken notwendig.

5.3.1. Strukturelle Datenintegrität

Unter der strukturellen Datenintegrität ist die Gewährleistung der im Relationalen Modell zur Verfügung stehenden Datenstrukturen zu verstehen. Einige, das ursprüngliche Konzept ergänzende Erweiterungen sind dabei aufzunehmen. Dies sind insbesondere die Ansätze von <Cod79>, <Dat81a>, <SS77>. Im Relationalen Datenmodell sind diese Eigenschaften durch den Aufbau der Relationen, die Normalformenlehre und Kriterien an die interrelationale Referenz von Relationen gegeben. Die wesentlichen Arten struktureller Integrität wurden bereits in Abs. 5.1.1 skizziert. Sie werden im folgenden weiter präzisiert, indem entsprechende Konstrukte zur Notation ins Konzeptionelle Schema eingebracht und ihre Kontrollsphären aufgezeigt werden.

Ein Relationenschema baut sich nach Abs. 1.1.2 in der in Abs. 2.2.2 eingeführten Notation wie folgt auf:

```
TYPE relation : RELATION OF RECORD
    attr1 : <type>;
    attr2 : <type>;
    :
    attrn : <type>;
END;
```

Wobei "relation" für den Namen der zu definierenden Relation, attr1,..., attrn für die Namen ihrer Attribute und <type> für einen Datentyp steht, durch den der Wertebereich des jeweiligen Attributs bestimmt wird.

Die angegebene Definition eines Relationenschemas ist zu ergänzen um Konstrukte bezüglich der strukturellen Integritätsarten (vgl. Abs. 5.1.1). Im einzelnen ergibt sich:

(1) Typintegrität

Jedem Attribut wird ein Wertebereich zugeordnet. In den gängigen Datenbanksystemen beschränkt sich die Angabe auf die in den höheren Programmiersprachen gebräuchlichen Angaben zum Datentyp (vgl. SQL/DS <IBM81>, INGRES <RT82>). Dies ist zu wenig, wünschenswert sind Angaben über spezifische benutzerdefinierte Datentypen (vgl. <McL76>, <Schm78>, <Thu80>, <Bro80>, <RRUZ82>, <Wel82>). Eine Personalnummer ist z.B. nicht einfach vom Typ "Integer" oder "String", sondern unterliegt ganz bestimmten anwendungsspezifischen Konventionen. Die Vereinbarung beliebiger Datentypen und die Einbettung einer entsprechenden Typprüfung sollte deshalb möglich sein. (Bei kommerziellen System wird als zusätzlicher Datentyp oft lediglich speziell die Datumsangabe unterstützt.) Die Notwendigkeit neuer Datentypen impliziert auch die Möglichkeit zur Definition von Ordnungsrelationen auf diesen Typen. Sie ist erforderlich für den ordnenden Vergleich in den relationalen Datenbank-Operationen, wenn man sich nicht auf den Test auf Gleichheit und Ungleichheit beschränken möchte.

Für die Notation einer strikten Angabe von Datentypen reicht die obige Notation für Relationen aus, wenn man die Möglichkeiten zur Definition von Typen analog zu Modula-2 (<Wir82>, S.61) in der Form von Aufzählungstypen oder über Formatvereinbarungen durch Angabe von Recordtypen aus bereits vorhandenen (elementaren) Datentypen zulässt. (Die Datentypvereinbarungen sind so vorzunehmen, daß die Normalformenlehre für den Aufbau von Relationen nicht verletzt wird.)

Beispiele für Aufzählungstypen finden sich in der Beispiel-Datenbank des Anhang 1.

z.B. **TYPE** Titel = (Dr.med, Dr.med.dent, PD Dr.med.habil.,
Prof.Dr.med, ...)

aus der Relation ARZT (Anhang 1).

Die Kontrollsphäre, die durch das explizite Festlegen des Datentyps aufgespannt wird, beschränkt sich auf das betroffene Attribut. Die Kontrolle läßt sich folglich über jeden (einzutragenden oder zu löschenden) Wert eines Tupels und die Definition des Datentyps erreichen. Diese Art der Integrität ist somit auf der Ebene der elementaren Datenbank-Operationen prüfbar. Es kann, wenn das Kommunikationssystem Kenntnis von den zulässigen Datentypen der Datenbank hat (z.B. durch ein gemeinsames Data Dictionary), die Prüfung bereits ohne Datenbankzugriff bei der Eingabe erfolgen.

(2) Schlüsselintegrität

Eine weitere Eigenschaft des Relationalen Modells ist die Identifikationseigenschaft einiger ausgezeichneten Attribute einer Relationen, für die einzelnen Tupel. Diese Attribute bezeichnet man als (Primär)-Schlüsselattribute.

Es gilt:

Ist $R(A_1, \dots, A_j, \dots, A_n)$ ein Relationenschema, dann ist $A_1, \dots, A_j$ genau dann ein Schlüssel bezüglich der Relation R , wenn gilt

$$\|R(A_1, \dots, A_n)\| = \|R(A_1, \dots, A_j)\|$$

Bei der Definition der Relationen ist diese Vereinbarung der Schlüssel explizit in einem Sprachkonstrukt anzugeben. In Anlehnung an die von Thurnherr <Thu80> eingeführte Notation ergibt sich folgende Erweiterung:

```
TYPE relation : RELATION OF RECORD
      attrl : <type>;
      :
      attrn : <type>;
      IDENT = (attrl,...,attri)
END;
```

Die durch die Integritätsbedingung auf gespannte Kontrollsphäre umfaßt in diesem Fall die Schlüsselattribute der Relation. Für die Prüfung ist es ausreichend, aufgrund der speziellen Schlüsseleigenschaft nur einmal auf die Relation zuzugreifen. Die Verletzung der Schlüsselintegrität kann somit ebenfalls auf der Basis elementarer Datenbank-Operationen erkannt und verhindert werden.

(3) referentielle Integrität

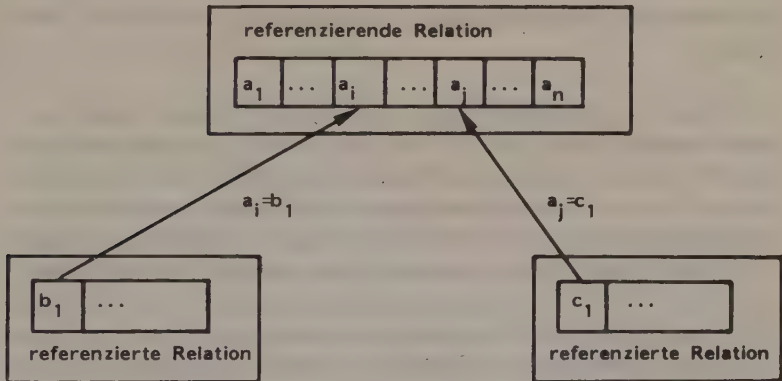
Die verschiedenen Arten referentieller Integrität, die in Abs. 5.1.1 eingeführt wurden, lassen sich im Hinblick auf die Kontrolle einheitlich behandeln. Die Notation für die referentielle Integrität wird wiederum aufbauend auf Thurnherr <Thu80> vorgenommen. Thurnherr faßt die referentielle Integrität als Angabe dynamischer Wertebereiche für die abhängigen Relationen auf. Gekennzeichnet werden sie über IMPORT und EXPORT Klauseln, wie sie in Modula-2 zwischen Moduln vorhanden sind.

Man kann die referentiellen Integritäten, da sie für Beziehungen zwischen Relationen stehen, entweder bei der abhängigen, referenzierenden Relation und eventuell zusätzlich bei der referenzierten Basisrelationen notieren (analog zu Thurnherr). Möglich ist auch die Integritätsbedingungen getrennt von den Relationen anzugeben. Im folgenden werden die referentiellen Integritätsbedingungen bei der Definition der referenzierenden Relation angegeben. Die Angabe des Datentyps eines Attributs wird dahingehend modifiziert, daß spe-

zifiziert wird, welches Attribut aus welcher Relation als "dynamischer" Wertebereich eingeht.

Abb. 5-4.:

(* Dynamische Wertebereichsangabe bei referentieller Integrität *)



Der Aufbau einer Relation "relation2" mit einem komplexen Schlüssel, der sich auf eine Basisrelation "relation1" bezieht, ist dann z.B. nach dem folgenden Schema zu definieren.

```

TYPE relation1 : RELATION OF RECORD
    attr11 : <type>;
    :
    attr1n : <type>;
    IDENT = (attr11)
END;
```



```
TYPE relation2 : RELATION OF RECORD
    attr21 : relation1.attr11;
    attr22 : relation1.attr11;
    :
    attr2n : <type>;
IDENT = (attr21,attr22)

END;
```

(Das Beispiel entspricht der referentiellen Integrität nach Codd.)

Alle referentiellen Integritätsarten lassen sich auf diese Weise darstellen, mit Ausnahme der Smith/Smith'schen <SS77> Hierarchisierung von Begriffen. Wenn durch ein Attribut in einer "Gattungsrelation" auf die "Artrelationen" verwiesen werden soll, hat als Typangabe dieses Attributs die Menge der Relationennamen zu stehen, die im Konzeptionellen Schema der Datenbank definiert sind. (Ist das Data Dictionary ebenfalls relational aufgebaut, dann gibt es eine Relation "RELATION" als Verzeichnis der im Konzeptionellen Schema enthaltenen Relationen, in der die Relationennamen Schlüssel sind. Diese Relation des Data Dictionary wäre dann zur Darstellung der Begriffshierarchie nach Smith/Smith <SS77> zu referenzieren.)

In die Angabe der Kontrollsphäre gehen bei der referentiellen Integrität sowohl die referenzierte wie auch die referenzierende Relation mit dem abgeleiteten Attribut ein. Man erhält eine interrelationale Beziehung *). Zur Integritätssphäre der referentiellen Bedingungen gehört bei der Manipulation eines der beiden in Beziehung gesetzten Attribute auch das jeweils andere. Bei der referentiellen Integrität

*) Die Bezeichnung von Integritätsbedingungen, die sich nur auf eine Relation beziehen, als intra-relational und diejenigen, die mehrere Relationen umfassen, als inter-relational geht auf Schmidt <Schm78> zurück.

handelt es sich um eine inter-relationale Bedingung , da sie mehrere Relationen betrifft.

Eine Verletzung der referentiellen Integrität kann im allgemeinen nicht mehr auf der Basis elementarer Datenbank-Operationen entschieden werden. Die referentielle Integrität muß erst auf höherem Niveau z.B. am Ende einer Aktion oder Transaktion gelten. Die Möglichkeit einer früheren Erkennung und Behandlung auf der Ebene der Datenbank-Anwendungsmoduln wird in Abs. 5.4 aufgezeigt.

Die referentiellen Integritätsbedingungen sind deshalb zu sammeln und nach der Methode der verzögerten Prüfung <EC75>, <Web81> beim Transaktionsende (oder in Erweiterung des Gedankens beim Aktionsende) zu behandeln. Verschiedene Implementierungsvorschläge hierzu werden von Weber <Web81> einander gegenüber gestellt.

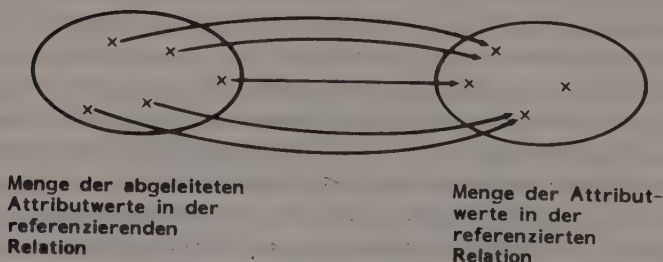
Die referentielle Integrität als datenmodellspezifische Integrität resultiert aus zwei Forderungen des Datenmodells. Im Relationalen Datenmodell sind alle Beziehungen zwischen den Relationen (oder allgemeiner Datensätzen) explizit darzustellen. Diese Forderung führt zum Einbringen von "Fremdschlüsseln" oder "dynamischen" Wertebereichen für Attribute <Thu80>, die in mehreren Relationen auftreten. Eine andere Forderung des Relationalen Datenmodells ist die Redundanzfreiheit. Es sollen keine Daten mehrfach in der Datenbank gespeichert werden, mit der Intention Fehler und Widersprüche zu vermeiden, die durch Abweichungen zwischen den originären und den abgeleiteten Daten auftreten können. Diese Forderung wird durch die explizite Darstellung von Zusammenhängen zwischen den Relationen verletzt. Die hierzu notwendige, aber redundante Angabe von Attributwerten gilt es über Integritätsbedingungen abzusichern. In einem integren Zustand der Datenbank haben sich die redundanten, abgeleiteten Attributwerte stets in Übereinstimmung mit ihren zugehörigen Basiswerten zu befinden.

Die angeführten Arten referentieller Integrität lassen sich dahingehend verallgemeinern, daß man prinzipiell redundante Daten in der Datenbank zuläßt, sie also nicht auf die eben dargestellten Fälle beschränkt. Die Speicherung redundanter Daten wird in konkreten Anwendungen häufig aus Gründen des Leistungsverhaltens gefordert. Man benötigt dann jedoch zur Gewährleistung der Widerspruchsfreiheit der redundanten Daten weitergehende Konstrukte zur Formulierung referentieller Integrität. Es sind dann zum Beispiel bei der Angabe der dynamischen Wertebereichsangaben eingebettete Funktionen ("Built-In Functions") zu verwenden.

Referentielle Integrität, als dynamischer Wertebereich verstanden, bedeutet, es gibt zu jedem Wert des abgeleiteten Attributs einen funktional bestimmten Attributwert in der referenzierten Relation. Diese Funktion (im einfachen Fall die Gleichheit) ist total auf den Attributwerten der referenzierenden Relation, jedoch nicht surjektiv auf den Werten der referenzierten Relation.

Abb. 5-5.:

(* funktionaler Zusammenhang zwischen referenzierendem und referenziertem Attribut *)



Aus diesen Eigenschaften der Beziehung zwischen den Attributwerten folgt für die referentielle Integrität,

- (1) das Löschen von Attributwerten in der referenzierenden Relation und
- (2) das Einfügen von Attributwerten in der referenzierten Relation

sind auf der Basis der elementaren Datenbank-Operationen prüfbar.

Die Forderung nach Redundanzfreiheit impliziert einen geringeren Aufwand an Integritätsprüfungen bei Änderungen auf der Datenbank. Andererseits verringert sich der Aufwand beim Auswerten der Datenbank, wenn man die Speicherung redundanter Daten gestattet und wenn man sie mittels referentieller Integrität widerspruchsfrei erhält. Die Entscheidung, welche Alternative gewählt wird, ist abhängig vom Profil der Anforderungen der Anwendung an die konkrete Datenbank.

Eine Formulierung von Triggern zum Ergänzen der referentiellen Integrität ist auf dem Niveau elementarer Datenbank-Operationen nur bedingt sinnvoll. Eine Verletzung durch das Einfügen von Tupeln in der referenzierenden Relation könnte ein Einfügen ergänzender Tupeln in der referenzierten Relation verursachen. Eine Verletzung durch das Löschen von Tupeln in der referenzierten Relation könnte ein Löschen in der referenzierenden Relation bedingen. Die Initiierung dieser integritätssichernden Regeln ist aber erst bei EOT sinnvoll. Nur aus einer konkreten Anwendung heraus ist zu entscheiden, ob ein solches Vorgehen angebracht ist. Generell kann auf diese Weise nicht verfahren werden.

Zusammenfassende Bemerkungen:

- Die einzelnen Arten struktureller Integrität sind nur bedingt bei der Ausführung der elementaren Datenbank-Operationen prüfbar.
- Die referentielle Integrität ist mit den aufgezeigten

- Ausnahmen nicht auf der Basis der Datenbank-Operationen zu prüfen.
- Die Prüfung der strukturellen Integrität hat spätestens bei Transaktionsende zu erfolgen. Es sind Zwischenlösungen denkbar, etwa die Prüfung auf der Basis der Aktionen der Datenbank-Anwendungsmoduln.

5.3.2. Semantische Datenintegrität

Semantische Datenintegrität umfaßt in der Literatur <FSW81> auch die Fälle von Integrität, die hier unter dem Begriff der strukturellen Datenintegrität behandelt wurden. Die Unterscheidung ist insofern zu rechtfertigen, daß bei der Wahl eines anderen Datenmodells, die im vorherigen Abschnitt betrachteten Integritätsarten nicht in dieser Form auftreten würden.

Die semantische Datenintegrität verfolgt das Ziel, die potentiell möglichen Zustände der Datenbank, wie sie durch das Konzeptionelle Schema und durch die strukturelle Datenintegrität festgelegt sind, auf jene einzuschränken, die in der Miniwelt der Anwendung tatsächlich auftreten können (vgl. Abs. 5.1.1).

Die semantische Datenintegrität ist somit reduzierbar auf die Notwendigkeit einen gegebenen Zustand der Datenbank mit möglichen (d.h. zu garantierenden) bzw. mit nicht möglichen (d.h. zu vermeidenden) Zuständen zu vergleichen.

Zur Angabe von Zustände kann man in einer Relationalen Datenbank auf die mengenorientierte Datenbank-Abfragesprache zurückgreifen. Die Notation semantischer Integritätsbedingungen besteht aus einem Vergleich zweier Tupelmengen oder Werte.

Die Überprüfung der semantischen Integritätsbedingungen ergibt sich wiederum durch die Zuordnung einer bool'schen Funktion, die wahr ist, wenn der Vergleich zutrifft. Die semantischen Integritätsbedingungen lassen sich in zwei Hauptgruppen *) einteilen,

- (Mengen-)Bedingungen,
die auf den Vergleich von Tupelmengen basieren;
- (Einzelwert-)Bedingungen,
die auf den Vergleich von einzelnen Werten der in der Datenbank verwendeten Datentypen aufbauen.

Die semantische Integrität einer Datenbank ist erfüllt, wenn alle Vergleiche, die durch semantischen Integritätsbedingungen angegeben sind, zutreffen.

(1) Mengenbedingungen

Die Mengenbedingungen lassen sich nach dem folgenden Schema aufbauen. Es wird eine Tupelmenge aus dem aktuellen Datenbankzustand (mit Hilfe einer Datenbank-Abfragesprache) ausgewählt, diese wird verglichen (bezüglich der Ordnungsrelation von Mengen "enthalten in") mit einer zweiten Tupelmenge. Diese kann entweder wiederum aus dem aktuellen Zustand der Datenbank abgeleitet sein, oder explizit angegeben werden. (Einen wichtigen Spezialfall bilden einelementige Mengen.)

Die Mengenbedingungen können in Anlehnung an QUEL (QUEL kennt nur intra-relationale Bedingungen in eingeschränkter Form <RT82>) und SEQUEL <Cha76>, in der folgenden Notation angegeben werden.

*) Die Einteilung in Mengen- und Einzelwertbedingungen erfolgt abweichend von <EC75> und <Web81>. Eswaran/Chamberlin unterscheiden die Bedingungen im Hinblick auf die betroffene Kontrollsphäre, während hier nach den Elementen, die in Beziehung gesetzt werden, differenziert wird.

Eine mögliche Erweiterung von QUEL liese sich (analog zu den CONDITIONS bei den (Trans-)Aktionen), wie folgt, vornehmen:

```
RANGE OF r1 IS relation1;
```

```
RANGE OF r2 IS relation2;
```

```
DEFINE
```

```
ASSERTION ON relation1, relation2 IS
```

```
  (r1.attr11, ..., r1.attr1n)
```

```
  WHERE <qualification>
```

```
  .*.
  (r2.attr21, ..., r2.attr2n)
```

```
  WHERE <qualification>
```

```
oder
```

```
DEFINE
```

```
ASSERTION ON relation1 IS
```

```
  (r1.attr11, ..., r1.attr1n)
```

```
  WHERE <qualification>
```

```
  .*.
  { t1, ..., tm }
```

```
  (* als eine Menge n-stelliger Tupel *)
```

```
oder (der Spezialfall)
```

```
DEFINE
```

```
ASSERTION ON relation1 IS
```

```
  (r1.attr11, ..., r1.attr1n)
```

```
  WHERE <qualification>
```

```
  .*.
  t1 (* als ein n-stelliges Tupel oder exakter { t1 } *)
```

wobei für ".*." die Mengenvergleichsoperationen "<", ">", "<=", ">=", "=", "!=" für "ist echte Teilmenge von", etc. eingesetzt werden können. In den Abfragen sind Mengenoperationen zulässig. Entsprechende Beispiele zur semantischen Integrität sind im Anhang 1 aufgeführt.

Die so formulierten Bedingungen können in einem aktuellen Datenbankzustand entweder erfüllt oder verletzt sein. Bei EOT einer Transaktion müssen alle Integritätsbedingungen gelten, die zu der Kontrollsphäre der Transaktion gehören.

(2) Einzelwertbedingungen

Neben den Vergleichen von Tupelmengen lassen sich auch semantischer Integritätsbedingungen auf der Basis von Einzelwerten (oder elementaren Werten) formulieren. Durch eine eingebettete Funktion ("Built-In Function") der Datenbank kann einer ausgewählten Tupelmenge ein Wert eines bestimmten Wertebereichs zugeordnet werden. Die Wertebereiche sind dabei nicht beschränkt auf die gängigen Datentypen "integer", "string", "real". Wenn man, wie in Abs. 5.3.1 gesehen, ein strenges Datentypkonzept dem Relationalen Datenbanksystem zugrunde legt, sind auch weitere Datentypen (z.B. anwendungsspezifische Aufzählungstypen) zulässig. Durch die Existenz einer entsprechenden Ordnungsrelation auf diesen Wertebereichen ergibt sich analog zu den Mengenbedingungen die Möglichkeit eines Vergleichs der Funktionswerte. Er stellt sich in folgender Form dar (die Möglichkeiten der Sprache QUEL wurden dabei wieder in analoger Weise erweitert).

```
RANGE OF r1 IS relation1;
RANGE OF r2 IS relation2;
DEFINE ASSERTION ON relation1, relation2 IS
    (FUNC (r1.attr11, ..., r1.attr1n)
        WHERE <qualification>)
    .*.
    (FUNC (r2.attr21, ..., r2.attr2n)
        WHERE <qualification>)
oder
DEFINE ASSERTION ON relation1 IS
    (FUNC (r1.attr11, ..., r1.attr1n)
        WHERE <qualification>)
    .*.
const
    (* als eine Konstante aus dem Wertebereich
       der Funktion FUNC *)
```

Der Ausdruck "FUNC" steht dabei für eine beliebige eingebettete Funktion des Datenbanksystems. Gängige, in Datenbanksysteme eingebettete Funktionen sind z.B. COUNT (Mächtigkeit einer qualifizierten Tupelmenge), MAX, MIN (Maximal- und Minimalwert), SUM (zur Addition) etc..

Das Zeichen *. steht wieder für die gängigen Vergleichsoperatoren der Zahlen oder Zeichenketten. Bezieht sich die Funktion auf einen der nicht gängigen, benutzerdefinierten Datentyp, so ist die Existenz einer entsprechenden Ordnungsrelation zu fordern. (Bei der Angabe von Aufzählungstypen wie z.B. in Modula-2 erfolgt die Festlegung der Ordnung durch die Reihenfolge in der Aufzählung bei der Definition.)

(Bei Sichten ist auch ein Vergleich zwischen dem Zustand bestimmter Objekte der Datenbank und Werten von Variablen sinnvoll. Insbesondere gilt dies für Variable, die in die Bildung der Sicht eingehen, vgl. Anhang 1. So kann z.B. die Integrität der Einträge in der Relation ARZTBESUCH vom Außendienstmitarbeiter abhängig sein, dessen Personalnummer die Sichtbildung bestimmt.)

Durch die semantische Datenintegrität werden die Wertebereiche von Attributen in Abhängigkeit von einem vorliegenden Zustand der Datenbank eingeengt. Durch die Einbeziehung von Eigenschaften beliebiger Objekte der Datenbank in die Festlegung von Wertebereichen wird eine situationsabhängige Definition der Wertebereiche für die betroffenen Attribute erreicht.

Die Prüfbarkeit von Verletzungen der semantischen Integrität wird ausführlich von Weber (<Web81>, S.32ff) dargestellt. Es ergeben sich im wesentlichen folgende Prüfungssituationen:

- Wenn die Bedingung auf ein Tupel beschränkt ist, ist sie prüfbar auf der Ebene der elementaren Datenbank-Opera-

tionen.

- Wenn die Bedingung intra-relational ist, jedoch mehrere Tupel umfaßt, ist sie u.U. prüfbar auf der Ebene der elementaren Datenbank-Operationen. Die Möglichkeit ist abhängig von Art des Datenbankzugriffs. Man kann durch Speicherung redundanter Information die Prüfung von semantischen Integritätsbedingungen mit eingebetteten Funktionen unterstützen. (Man speichert z.B. Durchschnittswerte relationenspezifisch ab. Sie werden bei Änderungen entsprechend mitgeführt. In einer Integritätsprüfung, in der ein Durchschnittswert eingeht, ist dieser damit nicht jeweils erneut zu berechnen. Es werden folglich auch keine Zugriffe zu den einzelnen Tupeln der Relation erforderlich.)
- Sind die Bedingung inter-relational, so sind sie auf der Ebene der elementaren Datenbank-Operationen nicht prüfbar. Dies gilt analog zur strukturellen Integrität. Eine sinnvolle Prüfung kann nur zurückgestellt werden und bei EOT oder dem Ende einer Aktion erfolgen.

Zusammenfassung:

- Die semantische Datenintegrität legt mittels beliebiger Prädikate über Objekte der Datenbank die Wertebereiche situationsabhängig fest.
- Die Prädikate lassen sich mit Hilfe der relationalen Datenbanksprache, ergänzt um Funktionen, formulieren.
- Die Bedingungen basieren auf dem Vergleich von Tupelmengen oder auf dem Vergleich von Werten, die durch eingebettete Funktionen erzeugt werden.

5.3.3. Finale deontische Integrität

Durch die finale deontische Integrität werden die Datenbankzustände in vier Klassen eingeteilt. Ein konkreter Datenbankzustand ist demnach entweder geboten, erlaubt, verboten oder indifferent. Ein Datenbankzustand ist in eine der vier Klassen einzuordnen. Die Beurteilung eines konkreten Zustands ist dabei situationsabhängig (vgl. Abs. 5.1), d.h., er ist abhängig vom vorausgegangenen, integeren Zustand.

Die Beurteilung bedingt in Abhängigkeit von der Klassenzugehörigkeit; ob und inwieweit Maßnahmen zur Integritätssicherung erforderlich werden. Zur Darstellung der finalen deontischen Integrität kann in den Grundkomponenten auf die Notation der semantischen Bedingungen zurückgegriffen werden. In die finale deontische Integrität gehen zwei Datenbankzustände ein, ein vorliegender, alter Zustand und ein indenterter, neuer Zustand. Die Bewertung hinsichtlich der deontischen Modi bezieht sich dabei auf den intendierten Zustand.

Es ist zu bemerken, daß die finale deontische Integrität zwar zustandsbeschreibend ist, indem über die Zulässigkeit von Datenbankzuständen entschieden wird; sie legt aber auch fest, welche Datenbankzustände aufeinander folgen dürfen. In der Literatur sind Ansätze zur Darstellung dieser Integritätsart bei Eswaran/Chamberlin <EC75> als "transition integrity" und bei Weber <Web81> als "Übergangsintegrität" zu finden. Es fehlt jedoch die explizite Differenzierung zur operationalen semantischen Integrität (vgl. Abs. 5.1.1), die die Korrektheit des spezifischen Übergangs ansieht (vgl. Abs. 5.5 und Kap. 4) mittels Vor- und Nachbedingungen festlegt.

Die finale deontische Integrität wird beschrieben durch invariante Bedingungen für die Transformation von Datenbankzuständen, die für alle Transaktionen gelten müssen.

In Abs. 5.1.2 wurden entsprechend den deontischen Modifizier Klassen von Zuständen aus der Handlungstheorie hergeleitet. Die verschiedenen Konsequenzen, die sich aus der Zugehörigkeit eines Datenbankzustands bezüglich der einzelnen Klassen ergeben, sind von der Integritätskontrolle zu beachten.

(1) Erlaubter Datenbankzustand

Ein Datenbankzustand soll dann als erlaubt gelten, wenn er nach den in der Miniwelt geltenden Normen zustande kam. Die Erlaubnis ist somit abhängig vom vorausgegangenen Zustand. In einem Datenbank-Anwendungssystem ist festzulegen, welche Zustände als explizit erlaubt zu betrachten sind.

Die Notation der finalen deontischen Integritätsbedingungen zur Definition erlaubter Zustände könnte in Anlehnung an QUEL <RT82> in der Form erfolgen:

DEFINE PERMISSION ON <relations>

(* Angabe der betroffenen Relationen *)

FOR TRANSITION

FROM

(<target-list> WHERE <qualification> .* . . .)

(* Beschreibung des Ausgangszustands der Datenbank *)

TO

(<target-list> WHERE <qualification> .* . . .)

(* Beschreibung des intendierten Zustands *)

Die Beschreibung der Ausgangszustände und der intendierten Zustände erfolgt analog zu den Zustandsbeschreibungen in den semantischen Integritätsbedingungen.

Ein Beispiel für eine explizite Erlaubnis ist aus der Pharmadatenbank: "Der Eintrag von mehr als 8 Arztbesuchen pro Arbeitstag ist erlaubt".

Die Angabe explizit erlaubter Zustände mag im ersten Augenblick als überflüssig erscheinen. Im Hinblick auf die im Kapitel 6 vorgenommene Individualisierung der Betrachtung bezogen auf einzelne Benutzergruppen, gewinnt die explizite Erlaubnis zur Modifikation von Daten an Bedeutung.

Ein anderer Aspekt expliziter Erlaubnis ist, daß bei offenen Systemen zwischen erlaubten und indifferenten Zuständen unterschieden werden kann. Beim Auftreten eines indifferenten Datenbankzustands können dann Transaktionen initiiert werden, die auf den Zustand aufmerksam machen oder korrigierend eingreifen. Im obigen Beispiel wäre die Benachrichtigung des Gruppenleiters, der die Gründe für den indifferenten Zustand (wenn der Mitarbeiter weniger als 8 Ärzte besucht) näher untersuchen kann.

(2) verbotener Datenbankzustand

Die Angabe finaler deontischer Integritätsbedingungen verhindert auch, daß verbotene, den Normen der Anwendung nicht entsprechende Datenbankzustände erzeugt werden. Wird ein solcher Übergang versucht, so hat das Kontrollsystem die Datenbank auf einen erlaubten (z.B. auf den vorher geltenden) konsistenten Zustand zu bringen (z.B. durch Zurücksetzen).

Die Notation der verbotenen Zustände kann man somit entsprechend zu den erlaubten Zuständen erfolgen:

DEFINE PROHIBITION ON <relations>

FOR TRANSITION

FROM

(<target-list> WHERE <qualification> *. ...)

TO

(<target-list> WHERE <qualification> *. ...)

Aus einer Pharmadatenbank als Beispiele hierzu: "Der Musterbestand eines Außendienstmitarbeiters darf einen bestimmten Wert nicht übersteigen" oder "Die Arztbesuchseinträge der vergangenen Woche dürfen nicht mehr verändert werden".

Die Reaktion auf verbotene Datenbankzustände kann differieren. Es ist zum einen möglich, die den Übergang vollziehende Transaktion zu verhindern oder zurückzusetzen. Zum anderen ist es möglich, den verbotenen Zustand durch Initiierung weiterer Transaktionen zu kompensieren. Beide Fälle können vom System unterstützt werden. Zum Beispiel kann das Eintreten eines verbotenen Zustands nur im mittelbaren Zusammenhang mit der Transaktion stehen, die in verursacht. Zum obigen Beispiel: "Der Preis für ein Medikament wird erhöht." Dies kann bewirken, daß die Wertgrenze der bei einem Mitarbeiter lagernden Medikamente überschritten wird. Es ist sicherlich nicht sinnvoll, die Transaktion zur Preiserhöhung aus diesem Grund scheitern zu lassen.

(3) gebotener Datenbankzustand

Die Betrachtung gebotener Datenbankzustände ist von Interesse, wenn man in einem vorliegenden Zustand die Erzeugung Zustand für gebotenen erachtet. Durch die Angabe gebotener Übergänge ist es dem Datenbank-Anwendungssystem möglich, die Erzeugung eines gebotenen Zustands durch die Ausführung geeigneter Transaktion zu erzwingen. Das Anwendungssystem kann dahingehend aktiv werden, daß es den Benutzer zum Handeln auffordert, wenn er auf Objekte aus der Sphäre der gebotenen Bedingung zugreifen will (vgl. Kap. 6).

Die Formulierung gebotener Übergänge ist die Begründung für die Möglichkeit und Notwendigkeit transaktionsorientierter Trigger, auf die im Abschnitt 5.5 noch eingegangen wird.

DEFINE OBLIGATION ON <relations>

FOR TRANSITION

FROM

(<target-list> WHERE <qualification> .* ...)

TO

(<target-list> WHERE <qualification> .* ...)

(Das Datenbank-Anwendungssystem kann den gebotenen Zustand auch automatisch erzeugen, wenn die hierfür notwendige Transaktion im Datenbanksystem vollständig verfügbar ist. Es liegt dann ein Fall einer afinalen deontischen Integritäts-sicherung vor, vgl. Abs. 5.5.1)

(4) indifferenter Datenbankzustand

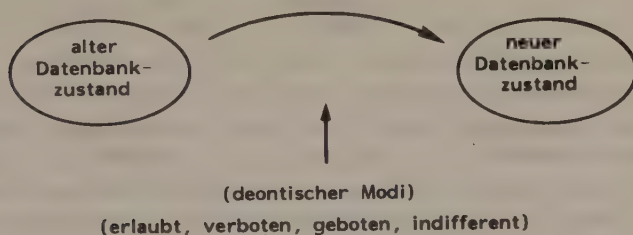
Wenn man für die Miniwelt ein offenes Normensystem zugrunde legt, ergeben sich indifferente Zustände, die sich einer Beurteilung hinsichtlich der obigen Kriterien entziehen. Es ist eine rein anwendungsspezifische Entscheidung, nach welchem Prinzip man in einer Datenbank-Anwendung oder Miniwelt verfahren möchte. Eine größere Flexibilität bietet in der Regel ein offenes System. In manchen Teilbereichen einer Miniwelt und dementsprechenden Sichten der Datenbank ist dennoch ein geschlossenes System vorzuziehen.

Zur deontischen Basis

Eine Datenbank befindet sich in einen deontisch integren Zustand, wenn alle finalen deontischen Bedingungen erfüllt sind. D.h. insbesondere auch, daß kein Zustand vorliegt, der einen gebotenen Nachfolgezustand fordert.

Abb. 5-6.:

(* Bewertung der Datenbankzustandsübergänge in den deontischen Modi *)



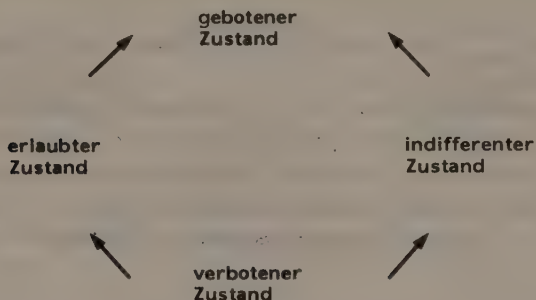
Die vier Klassen der möglichen Datenbankzustände bezüglich der finalen deontischen Integrität lassen sich zueinander in Beziehung setzen.

- gebotener Datenbankzustand,
die Erzeugung des Zustands wird aufgrund der Normen gefordert;
- erlaubter Datenbankzustand,
der Zustand ist aufgrund expliziter Erlaubnis erzeugt;
- indifferenter Datenbankzustand,
für den Zustand liegt weder eine explizite Erlaubnis, Verbot oder Gebot vor;
- verbotener Datenbankzustand,
der Zustand darf nicht erzeugt werden bzw. ist zu beseitigen.

Für die Kontrolle der finaler deontischer Integrität auf der Basis der Datenbank stellt sich die Frage, wie man die Bedingungen überwachen und die einzelnen Übergänge verhindern bzw. erzwingen kann.

Abb. 5-7.:

(* Beziehung zwischen den Datenbankzuständen bei finaler deontischer Integrität. *)



Durch die Angabe der finalen deontischen Integrität erhält man Bedingungen der folgenden Form:

int : DATABASE-OLD X DATABASE-NEW -----> { deontische Modi }

Die Bedingung charakterisiert die Zugehörigkeit eines zu erzeugenden Datenbankzustands "db-new" zu einer deontischen Klasse in Abhängigkeit vom vorhergehenden Datenbankzustand "db-old". Die Integritätssphäre spannt sich dabei über die betroffenen Attribute sowohl des alten wie auch des neuen Datenbankzustands auf.

Die Integritätsüberwachung hat im Gegensatz dazu zu beurteilen, welcher Klasse ein erzeugter Zustand zuzuordnen ist, um dann entsprechende Maßnahmen zur Integritätssicherung einzuleiten.

Es ergibt sich somit folgende Form:

int-cond : DATABASE-OLD X DATABASE-NEW X {deontische Modi}
 -----> { TRUE, FALSE }

Die zu Beginn dieses Abschnitts gegebene Notation der finalen deontischen Integritätsbedingungen (in der Erweiterung von QUEL) erfüllt diese Form.

Mögliche Reaktionen

Bei Verletzungen finaler deontischer Integrität ist eine Reaktion auf der Basis elementarer Datenbank-Operationen nicht möglich. Nach den deontischen Kriterien ist eine Handlung der Miniwelt zu beurteilen, die im Datenbank-Anwendungssystem durch eine Transaktion beschrieben wird. Die Behandlung der deontischen Integrität ist deshalb rein transaktionsspezifisch zu sehen. Entsprechende Möglichkeiten zur Kontrolle werden in Abs. 5.5 aufgezeigt.

Das Grundprinzip der Reaktion auf die Verletzung einer semantischen oder strukturellen Datenintegrität war, daß die Negation der Integritätsbedingung die "unerwünschten" Zustände beschreibt. Tritt ein Zustand aus diesem Komplement der Integritätsbedingung auf, kann spezifisch darauf reagiert werden. Bei der Behandlung der finalen deontischen Integrität kommt man aufgrund der Einteilung der Datenbankzustände anhand der deontischen Modi nicht mit einer einfachen Komplementbildung aus. Das Komplement zu einer deontischen Bedingungen umfaßt jeweils drei Klassen von Datenbankzuständen. Die Reaktion auf eine "Verletzung" ist abhängig von der Art der Integritätsbedingung und der Klassenzugehörigkeit des tatsächlich erzeugten Zustands.

Die gebotenen, erlaubten und indifferenten Datenbankzustände bilden die Klasse der zulässigen Zustände. Wird z.B. eine Integritätsbedingung verletzt, die einen erlaubten Zustandsübergang sichert, so ist die Reaktion davon abhängig, zu welcher Klasse der erzeugte "unerwünschte" Zustand gehört. Die Möglichkeiten hierzu sind z.B. das Rücksetzen beim Erreichen eines verbotenen Zustands oder das Melden des Erreichens eines indifferenten Datenbankzustands an einen

spezifischen Benutzer. (Wurde ein gebotener Zustand erreicht, ist keine Reaktion erforderlich.)

Zusammenfassung:

- Die finale deontische Integrität unterscheidet vier Klassen von Datenbankzuständen, die jeweils einer spezifischen Integritätsbehandlung bedürfen.
- Die finale deontische Integrität kontrolliert die Zulässigkeit der Transformation von Zuständen hinsichtlich der deontischen Modi.
- Die finale deontische Integrität ist nur transaktionsorientiert zu erfüllen; d.h., ihre Einhaltung kann nur auf der Basis der Transaktionen entschieden werden.

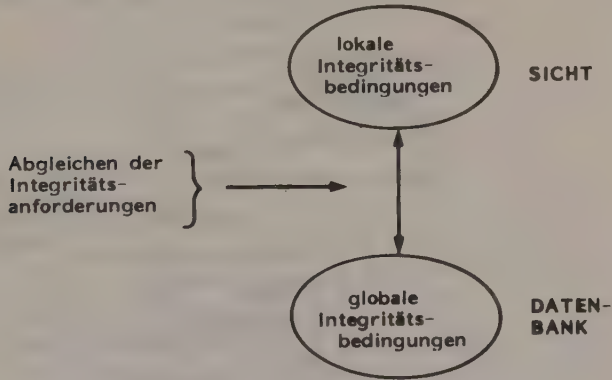
5.3.4. Globale und lokale Integrität - Beziehungen zwischen Datenbank und Sichten

Die strukturelle, semantische und finale deontische Datenintegrität wurde in den vorherigen Abschnitten auf der Ebene der Datenbank behandelt. Die gleichen Anforderungen ergeben sich aus der Architektur eines Datenbank-Anwendungssystems für die lokalen Sichten. Die Datenintegrität ist demzufolge auf zwei Ebenen des Anwendungssystem zu gewährleisten,

- global, bezüglich der Datenbank, und
- lokal, bezüglich der Datenbanksichten.

Abb. 5-8.:

(* Beziehung zwischen der Datenintegrität auf den statischen Sichten und der Datenbank *)

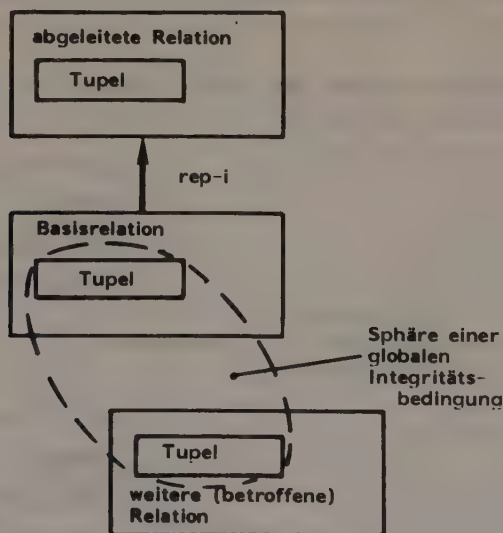


Eine Sicht ist ein lokaler Ausschnitt aus der Datenbank, der einer Benutzergruppe zur Verfügung gestellt wird (vgl. Abs. 3.1). Die Bildung der Sicht erfolgt mittels einer Sichtbildungsfunktion. Zwei Aspekte sind näher zu untersuchen:

- Ist es möglich und wünschenswert, Integritätsbedingungen auf der Sicht zusätzlich zu den Bedingungen auf der Datenbank zu formulieren. Dies würde ein restriktiveres Vorgehen bezüglich der Sicht als auf der Datenbank bedeuten.
- Wie ist es möglich, Integritätsverletzungen der Datenbank zu behandeln, wenn die Kontrollsphäre der verletzten Bedingungen nur teilweise in der Sicht transparent ist.

Abb. 5-9.:

(* Integritätsverletzung in der Datenbank ohne Verletzung der Sicht *)



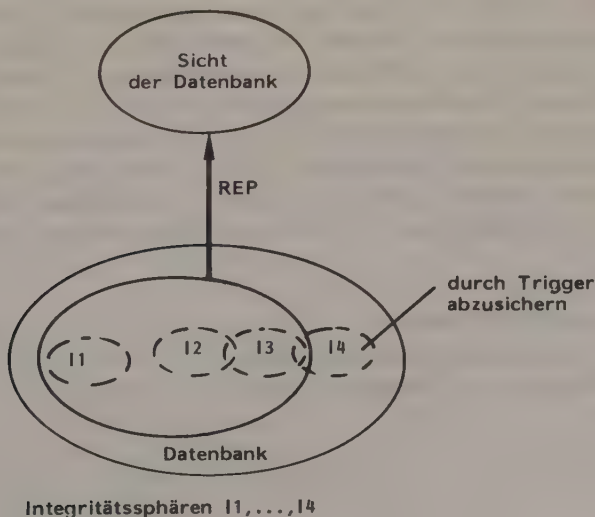
Als Beispiel für den zweiten Punkt, sei eine referentielle Integrität auf der Datenbank definiert, die durch eine Änderungsoperation der Sicht verletzt wird. Die referentielle Integrität betrifft zwei Relationen. Nur eine dieser Relationen geht in die Sicht ein, die andere ist dem Benutzer dagegen nicht transparent. Der Benutzer besitzt

auch keine weitere Möglichkeit diese verdeckte Relation zu ändern. Die durch die Integritätsbedingung betroffene Transaktion ist dann entweder zurückzusetzen oder es wird an die Sicht ein Trigger gebunden, der die Integritätsverletzung durch das Ausführen weiterer Datenbank-Operationen beseitigt.

Wenn die Integritätssphäre einer Änderungsoperationen auf der Sicht nicht die Sphäre auf der Datenbank umfaßt, so ergibt sich als eine Anforderung an die Bildung einer Sicht, daß ein Trigger in der Sicht (bzw. in dem zugehörigen Anwendungsmodul) vorzusehen ist, um das Scheitern von Transaktionen an dieser Bedingung zu verhindern. Bei der Bildung einer Sicht ist es folglich günstig, darauf zu achten, daß die Sicht bezüglich ihrer Integritätsbedingungen abgeschlossen ist. Die Basisrelationen der Sicht sollten keine weiteren Relationen der Datenbank referenzieren.

Abb. 5-10.:

(* Sicht und Integritätssphären *)



Prinzipiell ist es möglich, sichtspezifische lokale Integritätsbedingungen zu formulieren, um Operationen auf der Sicht restriktiver zu handhaben. Dadurch ergibt sich ein weiterer Gesichtspunkt für die Sichtbildung. Eine zulässige Transaktionsausführung liegt dann vor, wenn sowohl die Bedingungen der Sicht, wie die der Datenbank erfüllt sind.

Als Möglichkeit zur Prüfung bieten sich folgende Alternativen an:

- Es können die lokalen Bedingungen der Sicht in Bedingungen der Datenbank gewandelt werden. Die so entstandenen Bedingungen lassen sich dann konjunktiv mit den globalen Bedingungen der Datenbank verknüpfen. Die Schwierigkeit liegt in der Transformation der lokalen Bedingungen, da sie von der Sichtbildungsfunktion abhängig ist. Es treten damit ähnliche Anomalien auf, wie sie bei der Transformation von Änderungsoperationen in den Sichten in Operationen in der Datenbank (vgl. Abs. 3.1) gegeben sind.
- Es können die lokalen Bedingungen auf der Sicht und die globalen Bedingungen der Datenbank voneinander getrennt geprüft werden. Bei einer Überlappung der Integritätsbedingungen entsteht ein unnötiger Prüfaufwand. In diesem Fall sind deshalb, entweder die überflüssigen Integritätsbedingungen der Sicht oder der Datenbank zu eliminieren. Die Auswahl, welche Integritätsbedingungen zu streichen sind, ist anwendungsspezifisch. Die Gewährleistung der sichtspezifischen Integrität hat u.U. erhebliche Vorteile gegenüber einer globalen Prüfung, wie in <MS83> am Beispiel der Behandlung versionsbehafteter Objekte gezeigt wird.

Am Rande sei bemerkt, daß natürlich bei der Angabe lokaler sichtspezifischer Bedingungen die Konfliktfreiheit zu den Bedingungen der Datenbank gewährleistet sein muß.

5.4. Integritätskontrolle auf der Basis der Datenbank-Anwendungsmoduln

Auf der Ebene der Datenbank-Anwendungsmoduln werden die ersten anwendungsorientierten Aktionen spezifiziert (Abs. 3.2). Erst auf dieser Ebene lassen Aspekte der operationalen Integrität in puncto ihres strukturellen, semantischen (und deontischen) Aspekts behandeln. Die eigentlichen Träger der anwendungsbezogenen Handlungen sind jedoch die Transaktionen. Die Anforderungen der Datenintegrität, die während der Ausführung einer Transaktion verletzt sind, sind bereits teilweise in den Datenbank-Anwendungsmoduln prüfbar. In der operationalen Sicht ist auch zu berücksichtigen, inwieweit transaktionsspezifische Integritätsanforderungen bereits auf dieser Ebene der Datenbank-Anwendungsmoduln abgefangen werden können.

5.4.1. Möglichkeiten zur Kontrolle der Datenintegrität

Die Datenbank-Anwendungsmoduln beinhalten jeweils eine Sicht auf die Datenbank. Im zugehörigen Externen Schema sind die sichtspezifischen Integritätsbedingungen festgelegt, die bezogen auf diesen Ausschnitt aus der Datenbank zu gelten haben (vgl. Abs. 5.3.4). Die elementaren Datenbank-Änderungsoperationen verletzen die Datenintegritätsbedingungen (von der Typ- und Schlüsselintegrität abgesehen (vgl. Abs. 5.3.1)), da sie nur Änderungen auf jeweils einer Relation erlauben.

Nach dem Ansatz von Rebsamen/Zehnder (vgl. <RZ82> und Abs. 3.4) ist es auf der Ebene der Datenbank-Anwendungsmoduln möglich, eine Aktion aus einer Folge von Operationen so zu bilden, so daß keine Verletzung der lokalen, modulspezifischen Datenintegrität auftritt. Für die Transaktionsbildung genügt es also jene Integritätsbedingungen zu prüfen, die über die Aktionen eines Moduls hinausgehen. Die Problematik stellt sich damit analog zu der Beziehung zwischen Sicht und Datenbank (Abs. 5.3.4).

Die Frage ist, welche Arten der Integrität lassen sich auf der Basis der Datenbank-Anwendungsmoduln behandeln. Aus der Klasse der strukturellen Integrität ist die referentielle Integrität nicht auf der Basis elementarer Datenbank-Operationen kontrollierbar. Es bietet sich an, eine Folge von elementaren Datenbank-Operationen so zu konstruieren, daß die referentiellen Bedingungen erfüllt sind. (Die Problematik, daß referentielle Bedingungen durch die Art der Sichtbildung nicht im Datenbank-Anwendungsmodul transparent sind, sei ausgenommen).

Für die semantischen Integritätsbedingungen kann in analoger Weise vorgegangen werden. Die Aktionen werden so aufgebaut, daß die in einem semantischen Zusammenhang stehenden Objekte der Datenbank bei einer Änderung stets ge-

meinsam behandelt werden müssen. Die Voraussetzung dafür ist, daß eine Aktion integritätswahrend ist, d.h. sie erfüllt beim Aktionsende stets alle von ihr betroffenen Integritätsbedingungen.

Neben der aktionsspezifischen Überwachung wird auf der Ebene der Datenbank-Anwendungsmoduln auch eine modulspezifische Kontrolle vorgenommen. Sind semantische oder referentielle Integritätsbedingungen nach Ausführung einer Aktion verletzt, so besteht die Möglichkeit, einen modulspezifischen TRIGGER anzugeben. Durch die Triggerung einer ergänzenden Aktion innerhalb des Moduls wird die Sicht (und damit auch die Datenbank) in einen korrekten Zustand gebracht. Ein Datenbank-Anwendungsmodul ist so zu bilden, daß er als semantischen Bezug zur Miniwelt einer Menge von Objekten mit den zugehörigen Möglichkeiten des Handelns entspricht. Inkorrekte Zustände der Objekte können dann durch festgelegte ergänzende Handlungen einfach korrigiert werden.

Noch nicht betrachtet wurde die Möglichkeit einer Überwachung der finalen deontischen Integrität auf der Basis der Datenbank-Anwendungsmoduln. Die Kontrolle finaler deontischer Integrität auf der Basis von Aktionen würde bedeuten, daß ein Urteil in einem Ausgangs- oder Endzustand einer Aktion über die Zulässigkeit des Gesamtablaufs einer Transaktion möglich ist. Die Notwendigkeit für diese Prüfung erscheint aus Integritätsaspekten zweifelhaft. Sinnvoll kontrollierbar ist die deontische Integrität erst transaktions-spezifisch.

Bildung integritätswahrender Aktionen

Der Vorschlag, konsistenzhaltende Transaktionen in einem Datenbanksystem zu verwalten, geht auf Rebsamen/Zehnder <RZ82> zurück. Der Ansatz wurde in Abs. 3.4 bereits erörtert. Rebsamen/Zehnder fassen allerdings den Begriff der Konsistenz (oder Integrität) enger als es in dieser Arbeit geschieht. Sie beschränken sich im wesentlichen auf die strukturelle Integrität. Ihr Vorgehen ist deshalb auf Aktionen übertragbar.

(Von Rebsamen/Zehnder (<RZ82>, S.604) wird auch ein Vorschlag zu einer automatischen Generierung von integritätserhaltenden Aktionen gemacht. Diese Generierung umfaßt die Berücksichtigung der strukturellen Integritätsarten, wobei allerdings direkt auf den Relationen der Datenbank und nicht auf den Sichten aufgesetzt wird.)

Die Aktionen eines Datenbank-Anwendungsmoduls werden durch die Angabe von Vor- und Nachbedingungen (vgl. Abs. 2.2.2) spezifiziert. Diese Übergangsbedingungen sind zu ergänzen um die strukturellen und semantischen Integritätsbedingungen der Sicht (bzw. der Datenbank), die von der Aktion betroffen sind. Wie die betroffenen Integritätsbedingungen zu bestimmen sind, wurde in Abs. 5.2 aufgezeigt. Gelingt es, die Aktionen so zu realisieren, daß die invarianten Bedingungen der Sicht (und der Datenbank) erfüllt sind, wird eine Kontrolle der Integrität nach Aktionsausführung überflüssig. Die Zulässigkeit der Aktionen (und damit der die Aktionen benutzenden Transaktionen) ist dann mittels Verifikation zusicherbar.

Die Komplexität der Integritätsanforderungen erlaubt es u.U. nicht, die Aktionen stets integritätserhaltend aufzubauen. In diesem Falle kann aber durch die skizzierte Vorgehensweise bestimmt werden, welche invarianten Bedingungen der strukturellen oder semantischen Integrität durch eine

Aktion verletzt werden. Diese so bestimmten Integritätsbedingungen werden mittels Trigger modulspezifisch abgesichert. Der Trigger wird ausgelöst durch die verletzte Bedingung. Er kann aktiviert werden sofort nach der Ausführung der Aktion oder seine Initiierung wird bis zum Ende der Transaktion zurückgestellt.

Aus diesen Überlegungen ergibt sich, daß sich die betroffenen Integritätsbedingungen einer Aktion aus drei disjunkten Klassen bezüglich der Kontrollierbarkeit zusammensetzen.

$INT(action) =$

$INT-ACT(action) \cup INT-MOD(action) \cup INT-TRANS(action)$

Die aktionsspezifischen Bedingungen $INT-ACT(action)$ werden von der Aktion erfüllt. Die modulspezifischen Bedingungen $INT-MOD(action)$ werden von der Aktion verletzt, sie sind jedoch bei EOT mittels Triggerung modulspezifisch erfüllbar. Die transaktionsspezifischen Bedingungen $INT-TRANS(action)$ sind auch modulspezifisch nicht erfüllbar.

Wenn sich eine Integritätsbedingung modulspezifisch prüfen läßt, ergibt sich eine Vereinfachung des Prüfungsvorgangs. Die modulspezifische Prüfung wird auf die betroffenen Objekte beschränkt, die durch die Kontrollsphären der Aktionen aufgespannt werden (vgl. aktionsspezifische Kontrollsphäre Abs. 2.2 und Kap. 3). Eine allgemeine Prüfung auf der Basis der Datenbank bedeutet hingegen eine umfangreichere Prüfung, in der auch nicht betroffene Einträge aus dem Definitionsbereich einer Integritätsbedingung in die Prüfung mit einbezogen werden.

Sprachkonstrukte zur Formulierung integritätssichernder Regeln

Zur Formulierung der Überwachung von invarianten Bedingungen auf der Ebene der Moduln sind diese in der Spezifikation eines Moduls anzugeben.

Es ergibt sich für die Spezifikation folgende Erweiterung:

```
DB-APPLICATION-MODULE appl-module (<parameter-list>)
```

```
EXPORT ...
```

```
:
```

```
TRIGGER trigger-name
```

```
    ON <condition> THEN <action>;
```

```
:
```

```
ACTION action-name (<parameter-list>)
```

```
:
```

```
MODULE-END appl-module.
```

Die Trigger-Bedingungen ergeben sich aus den nicht erfüllten modulspezifischen Integritätsbedingungen INT-MOD der Aktionen des Moduls. Zur Bestimmung der welche Integritätsbedingungen von den Aktionen eines Anwendungsmoduls bereits erfüllt werden, ergänzt man die Aktionen in den Vorbedingungen um die Integritätsbedingungen. Man erhält so durch die Nachbedingung eine Aussage über die Gültigkeit dieser Bedingungen. In der Angabe der Nachbedingung sind die Integritätsbedingungen aufzuteilen in einen Anteil, der durch die Aktion erfüllt wird, und einen Anteil, der nicht erfüllt wird.

Die vollständige Spezifikation erweitert um die invarianten Bedingungen der Sicht (d.h. um die modulspezifischen Bedingungen) stellt sich wie folgt dar.

ACTION action-name (<parameter-list>)

PRE-CONDITION:

<action-predicate>

AND <integrity predicate>

(* die aktionsspezifische Vorbedingung wird um die betroffenen modulspezifischen Integritätsbedingungen erweitert *)

POST-CONDITION:

<action-predicate>

AND <integrity predicate part1>

(* die betroffenen Integritätsbedingungen werden teilweise durch die Aktion erfüllt (Teil1) *)

AND NOT <integrity predicate part2>

(* und teilweise verletzt (Teil2) *)

END-ACTION action-name.

Für den Aufbau der modulspezifischen Trigger ergibt sich dann, daß die von den Aktionen verletzten Integritätsbedingungen die Bedingungen zum Anstoß der Trigger bilden. Zusätzlich ist der Zeitpunkt der Initiierung anzugeben z.B. beim Ende einer Transaktion.

Man kommt dann zu folgender Form:

TRIGGER trigger-name

ON NOT <integrity predicate part2>

AND REQUEST(EOT-PREPARE)

(* für die nicht erfüllten Integritätsbedingungen der Aktionen des Moduls werden bei EOT die Trigger ausgelöst *)

THEN ACTION

PRE-CONDITION:

NOT <integrity predicate part2>

POST-CONDITION:

<integrity predicate part2>

TRIGGER-END trigger-name.

Reduktion des Prüfaufwands durch Aktionen

Wenn die Integritätsbedingungen, soweit dies möglich ist, in die Verifikation der Aktionen mit aufgenommen und als invariant nachgewiesen werden, so entfällt ihre Prüfung am Transaktionsende. Das heißt der Prüfaufwand wandert in die korrekte Implementierung der Aktionen bezüglich ihrer Spezifikation. Der Aufwand während der Transaktionsausführung, der durch Prüfmaßnahmen entsteht wird jedoch drastisch reduziert. Bei einer günstigen Wahl der Aktionen und durch die Angabe modulspezifischer Trigger kann erreicht werden, daß die Prüfungen auf dem Niveau der Datenbank überflüssig werden. Ausgenommen sind Einschränkungen, die sich aus der Repräsentationsfunktion zwischen Datenbank und Sicht ergeben (vgl. Abs. 5.2).

5.4.2. Möglichkeiten zur Kontrolle operationaler Integrität

Neben der Frage, was an Datenintegritätsbedingungen auf der Ebene der Datenbank-Anwendungsmoduln bereits kontrollierbar ist, ist von Interesse, inwieweit man Anforderungen der operationalen Integrität durch Mechanismen auf dieser Ebene unterstützen kann bzw. muß.

Für die Transaktion wurden im Abs. 5.1 völlig analog zur Datenbank drei Arten operationaler Integrität skizziert, die strukturellen Eigenschaften, die Korrektheit des Übergangs und die deontischen Kriterien. Die deontischen Kriterien können von den Überlegungen auf der Basis der Anwendungsmoduln ausgenommen werden. Sie sind nur für die Festlegung der Datenbankübergänge auf der Basis der Transaktionen von Interesse.

An dieser Stelle kann man sich auf die Betrachtung der strukturellen operationalen Integrität (in Abs. 5.1.1 auch als Ablaufintegrität bezeichnet) und der semantischen operationalen Integrität (in Abs. 5.1.1 auch als Übergangsintegrität bezeichnet) beschränken.

Zur Ablaufintegrität

Die strukturellen Eigenschaften für den Ablauf einer Transaktion sind die Isolation, die Atomarität und die Dauerhaftigkeit. Die Eigenschaft der Isolation wurde in der Arbeit ausgeklammert, da hierfür vielfältige Lösungen auf dem Niveau der Datenbank existieren <BHR80>, <GM82>. (Eine Erweiterung des Transaktionskonzepts könnte durch die Aufgabe des Kriteriums der vollständigen Isolation geschehen. Man könnte eine "semantische" Synchronisation auf der Basis der Datenbank-Anwendungsmoduln entsprechend dem Ansatz von <Ker82> vornehmen. Das implizite Teilen von Objekten der Moduln über die Datenbank wäre dabei noch zu berücksichti-

gen. Diese Vorgehensweise würde es auch erlauben kooperierende und konkurrierende Transaktionen in einem Datenbank-Anwendungssystem zuzulassen.)

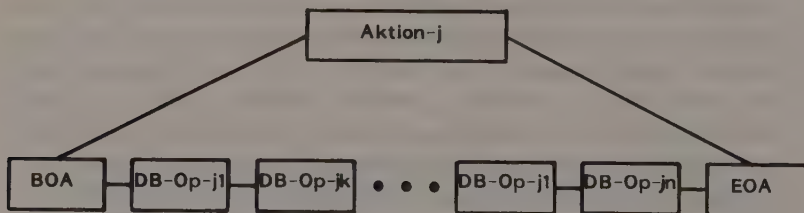
Die Forderung nach Atomarität stellt sich in zweifacher Form

- Atomarität der Aktionen und
- Atomarität der Transaktionen.

Die Ausführung einer Aktion hat atomar für die operationale Sicht zu sein, ebenso wie die Atomarität der elementaren Datenbank-Operationen angenommen wird. Die Atomarität der Aktionsausführung ist entsprechend der Konzeption der Transaktion vom Datenbankverwaltungssystem zu übernehmen. Die Implementation einer Aktion wird deshalb mit dem Datenbank-Kommando BEGIN-OF-ACTION eingeleitet und mit dem Kommando END-OF-ACTION abgeschlossen (vgl. Abs. 2.2.2). Im Fehlerfalle wird der Datenbankzustand zu BEGIN-OF-ACTION wiederhergestellt. Durch diese Eigenschaft kann auf der Ebene der Transaktionen, beim Scheitern einer Aktion, entschieden werden, ob die Transaktion abubrechen ist oder mit der Aktivierung einer alternativen Aktion fortgesetzt werden soll.

Abb. 5-11.:

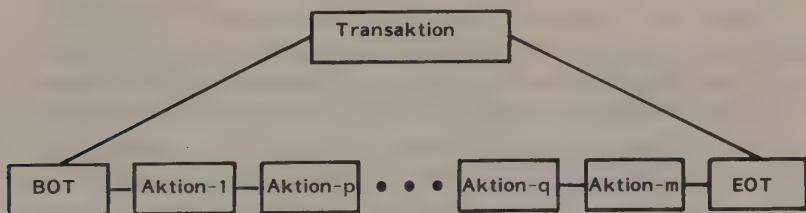
(* Atomarität von Aktionen *)



Die Atomarität der Transaktion erfolgt wiederum durch die Klammerung der Aktionen mit den Operationen BOT und EOT (vgl. Abs. 2.2.3). Die BOT- und EOT-Operation ist ebenfalls vom Datenbankverwaltungssystem zu interpretieren. Beim Abbruch der Transaktion ist die Datenbank auf den Zustand des BOT zurückzusetzen. Die Bereitstellung der BOT- und EOT-Operationen auf dem Transaktionsniveau kann auf zwei Arten erfolgen, entweder durch einen separaten Modul oder integriert in jedem Datenbank-Anwendungsmodul.

Abb. 5-12.:

(* Atomarität von Transaktionen *)



Die zweite Lösung hat mehrere Vorteile. Jeder Modul muß erkennen, daß eine Transaktion begonnen wurde, zu welcher Transaktion die jeweils ausgeführte Aktion gehört und wann die Transaktion beendet ist. Die Prüfmechanismen zur modulspezifischen Integritätskontrolle benötigen ebenfalls das EOT, damit sie aktiviert werden.

Die Vorgehensweise der Initiierung, der von einer Transaktion benutzten Modulen wurde bereits in Abb. 3-10 dargestellt. Die Möglichkeiten einer modulspezifischen Abschlußbehandlung von Transaktionen wurde ebenfalls bereits in Abs.3.3 skizziert.

Durch die BOT-Operation einer Transaktion werden BOT Aufrufe an alle von der Transaktion importierten Datenbank-Anwendungsmoduln abgesetzt. Die Anwendungsmoduln geben die Anmeldung an die Datenbank weiter und aktivieren ihre transaktionspezifische Kontrolle. Die EOT-Operation bewirkt ebenfalls EOT Aufrufe an alle importierten Datenbank-Anwendungsmoduln. Die Abwicklung der Aufrufe hat, wie in Abs. 3.3 beschrieben wurde, nach einem zwei-phasigen Protokoll zu erfolgen. Die modulspezifischen Trigger und die zurückgestellten Integritätsprüfungen haben nach der Phase des "Collecting" vor der Meldung "Bereit für Commitment" durch den Modul zu erfolgen.

Der dritte Aspekt struktureller operationaler Integrität war die Gewährleistung der Dauerhaftigkeit. Die Dauerhaftigkeit bedeutet einmal durch eine Transaktion freigebene (committed) Zustände können nur mittels Kompensation (durch die Ausführung einer "inversen" Transaktion) rückgängig gemacht werden. Dieses Strukturkriterium einer Transaktion ist auf der Ebene der Datenbank-Anwendungsmoduln nicht kontrollierbar. Da zu jedem Zeitpunkt der Transaktionsausführung der Abbruch erfolgen kann, ist es deshalb nicht sinnvoll durch Aktionen erzeugte Zustände anderen Transaktionen freizugeben.

Eine gewisse Notwendigkeit der Freigabe von Datenbankzuständen (Commitment) auf der Basis von Subeinheiten (in diesem Fall Aktionen) ergibt sich, wenn man sog. geschachtelte Transaktionen ("Nested Transaction", <Gra81>) zulässt. Für die Behandlung dieses Problems zeigt sich noch keine praktikable Lösung ab.

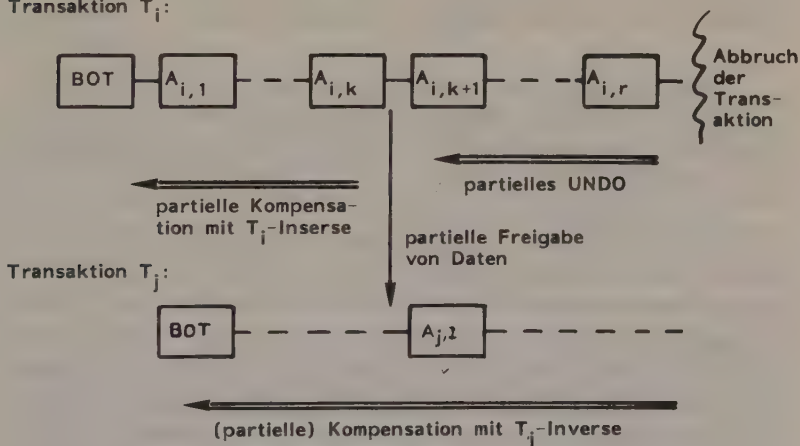
Eine Lösungsmöglichkeit im Rahmen der gemachten Überlegungen wäre eine aktionsspezifische (als Subeinheiten einer Transaktion) Freigabe zuzulassen. Im Falle des Scheitern der Transaktion, wäre ein Mechanismus zur Kompensation aller Transaktionen erforderlich, die den Effekt der Aktion refe-

renziert haben. Es wäre sicher zu stellen, daß nur Transaktionen auf den erzeugten Zustand zugreifen, für die auch eine inverse kompensierende Transaktion aktivierbar ist. Die Transaktion, die partielle Freigabe macht, kann dann aber ebenfalls nur bis zum letzten Freigabepunkt zurückgesetzt werden. Der zuvorliegende Teil der Transaktion wäre dann ebenfalls durch eine geeignete inverse (Rest-)Transaktion zu kompensieren.

Abb. 5-13.:

(* partielle Freigabe und Kompensation *)

Transaktion T_i :



Die skizzierte Vorgehensweise (Abb. 5-13) bedeutet eine Abschwächung der Forderung nach Isolation der Transaktionen. Es werden zwar keine direkten Wechselwirkungen während der Ausführung einer Transaktion sichtbar, jedoch wird eine Abhängigkeit der Transaktionen über die Notwendigkeit ihrer Kompensation aufgebaut.

Neben der Komplexität des Ansatzes ist seine Anwendung auch durch die Notwendigkeit die Spezifikation und Implementierung von Inversen Transaktionen beschränkt, da nicht jede Transaktion (vgl. <Gra81>) kompensierbar ist.

Zur Übergangsintegrität

Es erscheint eigentlich nicht sinnvoll, Teile der Spezifikation des Übergangs, der durch eine Transaktion festgelegt ist, auf der Basis der Anwendungsmoduln zu diskutieren. Auf der Ebene der Anwendungsmoduln sind die invarianten Bedingungen der Datenintegrität bezüglich einer Transaktion teilweise kontrollierbar, die transaktionspezifischen Vor- und Nachbedingungen jedoch nicht.

Die Vor- und Nachbedingungen der Aktionen zur Spezifikation des lokalen Übergangs wurden bereits in Abs. 2.2.2 ausführlich behandelt und in Abs. 5.4.1 um die Bedingungen zur Datenintegrität ergänzt.

Die Übergangsintegrität freier Transaktionen wird in ihrer Spezifikation lediglich durch die invarianten Bedingungen der Datenbank bestimmt. Wird zur Bildung von freien Transaktionen eine operationale Sicht zur Verfügung gestellt, ergeben sich die möglichen vollziehbaren Übergänge aus der transitiven Hülle ihrer Aktionen. Es wird in diesem Fall eine weitere implizite Einengung der möglichen Übergänge vorgenommen, die man als transaktionsbezogene Übergangsintegrität auf der Basis von Anwendungsmoduln interpretieren könnte.

5.5. Transaktionsorientierte Integritätskontrolle

Aus der Ebene der Transaktionen sind sowohl die operationalen Anforderungen wie auch alle Integritätsbedingungen über den Zustand der Datenbank zu erfüllen. Durch die Klassifikation der Transaktionen in schematische, reglementierte und freie ist es möglich die Kontrolle der Integrität durch abgestufte Mechanismen zu gewährleisten. Bereits bei der Behandlung der Aktionen auf der Basis der Anwendungsmoduln wurde deutlich, daß durch eine apriori Erkennung eine Integritätsverletzung teilweise verhindert werden kann. Auf die Transaktionen übertragen bedeutet dies, daß je schematischer der Transaktionsablauf bestimmt ist, desto früherzeitiger ist eine Erkennung einer Integritätsverletzung möglich.

Die Klasse der reglementierten Transaktionen erlaubt weitergehende Mechanismen zur Integritätskontrolle anzuwenden, als dies bei einer Kontrolle auf der Basis der freien Verwendung der einer Datenbanksprache der Fall ist. Reglementierte Transaktionen ermöglichen aber zusätzlich Handlungen der Miniwelt darzustellen, die sich nicht auf rein schematische Abläufe beschränken (vgl. Abs. 4.3).

Viele Mechanismen der Kontrolle sind den verschiedenen Transaktionsarten gemeinsam, sie werden im folgenden dargestellt. Die Unterschiede zwischen den einzelnen Transaktionsarten bezüglich der Integritätskontrolle werden anschließend behandelt.

5.5.1. Allgemeine Prinzipien transaktionsorientierter Integritätskontrolle

Die Kontrolle der Integrität wird bestimmt durch die Möglichkeiten, die sich zu Beginn und am Ende einer Transaktionsausführung ergeben, eine Verletzung zu beseitigen. Die Bedingungen bezüglich der Integritätsarten, wie sie in Abs. 5.1 aufgezeigt wurden, müssen zu diesen beiden Zeitpunkten erfüllt sein.

In diesem Abschnitt wird skizziert, welche Integritätsbedingungen zu Beginn und welche am Ende einer Transaktion auf ihre Gültigkeit zu prüfen sind. Daneben ist herauszuarbeiten, inwieweit für die einzelnen Transaktionsarten allgemeine Prinzipien der Kontrolle gültig sind.

Integritätsanforderungen zu Beginn einer Transaktion

Die Ausführung der BOT-Operation setzt voraus, daß sich die Datenbank in einem integeren Zustand bezüglich der semantischen und strukturellen Datenintegrität befindet. Diese Voraussetzung ergibt sich aus folgenden Eigenschaften:

- (1) Der Initialzustand einer Datenbank bei ihrer Installation ist integer.
- (2) Alle Transaktionen erzeugen aus einem integeren Datenbankzustand stets wiederum einen integeren.
- (3) Zwischenzustände der Datenbank, die während der Ausführung einer Transaktion entstehen, werden für andere Transaktionen nicht sichtbar bzw. nicht freigegeben.

Eine explizite Prüfung, ob bei BOT ein integerer Datenbankzustand vorliegt, entfällt aufgrund dieser Regeln.

Die Integritätskontrolle für den Transaktionsablaufs wird bei BOT, durch entsprechende Aufrufe an die Anwendungsmoduln eingeleitet, wie bereits in Abs. 2.2.3 und 5.4.2 er-

läutert wurde. Das Datenverwaltungssystem trägt durch entsprechende Vorkehrungen der Protokollierung dafür Sorge, daß die Transaktion jederzeit auf den Zustand bei BOT zurücksetzbar ist (vgl. <Gra77>). Die BOT-Operation schreibt damit einen ersten logischen Stützpunkt bezüglich der Datenbank, auf den beim Abbruch der Transaktion wieder aufgesetzt wird.

Zu Beginn einer Transaktion ist auch zu klären, ob die transaktionsspezifischen Vorbedingungen aus der Spezifikation erfüllt sind. Die Vorbedingung ist deshalb bei der Implementation einer Transaktion (in Form einer IF-Anweisung) als erstes zu prüfen (vgl. Kap. 4). Ist die Vorbedingung nicht erfüllt, erfolgt ein UNDO der Transaktion auf den Anfangszustand bei BOT.

Die aufgezählten Arten der Integrität sind zu ergänzen um die Vorkehrungen zur Prüfung der finalen und afinalen deontischen Integritätsanforderungen.

Kontrolle der afinalen deontischen Integrität

Die afinale deontische Integrität legt fest, ob die Ausführung einer Transaktion erlaubt, geboten oder verboten ist. Neben der Vorbedingung der Transaktion, die die Voraussetzungen für den Übergang beschreibt, ist eine Initialbedingung anzugeben, die festlegt unter welchen Situationen die Transaktion initiiert werden darf bzw. soll.

Die Spezifikation der Transaktionen erweitert sich dadurch (vgl. Abs. 4.2):

```
TRANSACTION transaction;  
SPECIFICATION;  
:  
CONDITIONS:  
  INIT-CONDITION:  
    TRIGGER ON <predicate>;
```

```
        PERMIT      ON <predicate>;
        REJECT      ON <predicate>;
PRE-CONDITION:
:
POST-CONDITION:
:
END transaction.
```

Die Erweiterung der Spezifikation um die drei zustandsbeschreibenden Prädikate der Initialbedingung (INIT-CONDITION) bestimmt, wie in einer gegebenen Situation die Initiierung der Transaktion deontisch zu beurteilen ist.

Ist das Prädikat in der TIGGER ON - Klausel erfüllt, ist die Transaktion umgehend auszuführen. Sie ist vom Datenbank-Anwendungssystem automatisch zu initiieren. Der "Benutzer", soweit er zur Ausführung überhaupt erforderlich ist, wird vom System zum Handeln aufgefordert. Die Triggerung von Transaktionen entspricht den gebotenen Handlungen, wie sie in der Handlungstheorie diskutiert werden (vgl. Abs. 5.1.2.2). Ein typisches Beispiel für eine zu triggernde Transaktion ist im Rahmen der Pharma-Datenbank die Erzeugung von statistischen Berichten, z.B. über den Besuchsschnitt der einzelnen Außendienstmitarbeiter bezüglich besuchter Ärzteguppen.

Die Beziehung zur finalen deontischen Integrität ist insofern herzustellen, daß gebotene Transaktionen bei ihrer Aktualisierung einen im Sinne der finalen deontischen Integrität gebotenen Zustand erzeugen (vgl. Abs. 5.3.3).

Ist das Prädikat der PERMIT ON - Klausel erfüllt, ist die Ausführung der Transaktion explizit erlaubt. Diese Klausel erhält ihre Bedeutung bei der personenbezogenen Betrachtung (Abs. 6.2).

Die Klausel REJECT ON spezifiziert alle jene Zustände, in denen eine Ausführung der Transaktion verboten ist. Eine Aktivierung der Transaktion wird dann vom Kontrollsystem verhindert *).

Für alle drei Initialbedingungen gilt: Die Prüfung der finalen deontischen Bedingungen nach Ausführung der Transaktion ist überflüssig, wenn die Initialbedingungen entsprechend der spezifizierten Wirkung der Transaktion mit den finalen deontischen Integritätsbedingungen verträglich sind. Da die Wirkung bei den reglementierten und freien Transaktionen nur teilweise apriori bestimmt ist, sind in diesen Fällen u.U. ergänzende Maßnahmen erforderlich (vgl. Abs. 5.5.2).

In der Implementation der Transaktion ist spezifisch auf den Versuch zur Aktivierung der Transaktion zu reagieren, wenn ihre Ausführung verboten oder nicht erlaubt (indifferent) ist. Die Triggerung einer Transaktion erfordert, daß dem Datenbank-Anwendungssystem die TRIGGER-Bedingung global bekannt gemacht wird. Bei einer Änderung des Datenbankzustands, die die TRIGGER-Bedingung betrifft, ist sicherzustellen, daß diese geprüft wird. Dies erfordert die Verwaltung der Trigger-Bedingungen an globaler Stelle (z.B. in einem aktiven Data Dictionary) und ihre Prüfung nach jedem EOT einer Transaktion.

Der Implementationsrahmen der Transaktionen erweitert sich zu Kap. 4 dahingehend,

*) Bemerkung: (Die Klauseln PERMIT und REJECT haben ihre Entsprechung in den Blockierungs- und Nichtblockierungsbedingungen, wie sie Keramidis <Ker82> zur Spezifikation von Anforderungen zur Synchronisation verwendet. Der Unterschied besteht nicht in der Darstellung, sondern in dem verfolgten Ziel. In dem vorliegenden Fall werden die Vorschriften zur Ausführung von Handlungen aus der Miniwelt nachgebildet.)


```
TRANSACTION transaction.  
IMPLEMENTATION;  
SCOPE;  
:  
:  
BEGIN-OF-TRANSACTION;  
:  
  IF <reject-condition>  
    THEN <action> UNDO;  
  IF NOT <permit-condition> THEN <action>;  
:  
END transaction.
```

Kontrolle der finalen deontischen Integrität

Die finale deontische Integrität beurteilt den Übergang zwischen zwei Datenbankzuständen. Zu Beginn einer Transaktion ist von Interesse, die Integritätsbedingungen zu finden, die nach der Ausführung zu prüfen sind. Eine erste Auswahl der betroffenen Bedingungen erfolgt über die importierten Datenbank-Anwendungsmoduln und den dort wiederum importierten Relationen.

Integritätsanforderungen am Ende einer Transaktion

Am Ende einer Transaktion müssen die Integritätsanforderungen erfüllt sein. Die Notwendigkeit, welche Bedingungen explizit zu prüfen sind oder inwieweit sie bereits durch eine korrekte Implementation der Transaktion erfüllt sind, ist von der Art der ausgeführten Transaktion abhängig.

Allgemein ist zu fordern,

- die semantische und strukturelle Datenintegrität muß bei EOT erfüllt sein,
- im Rahmen der finalen deontischen Integrität darf kein verbotener Zustand erzeugt werden, ansonsten hat das Datenbank-Anwendungssystem umgehend darauf zu reagieren.
- die Ausführung der EOT-Operation gibt die von der Transak-

tion benutzten Objekte der Datenbank frei und schließt die Transaktion als atomare Einheit ab.

- der vollzogene Übergang entspricht der in der Spezifikation festgelegten Datenbankzustandstransformation (die Notwendigkeit einer Prüfung ergibt sich spezifisch zur Transaktionsart),
- die afinale deontische Integrität ist mit der finalen abzugleichen. (Auch in diesem Fall ist die Notwendigkeit einer Prüfung von der Art der Transaktion abhängig).

5.5.2. Unterschiede in der Integritätskontrolle bei den verschiedenen Transaktionsarten

Die einzelnen Transaktionsarten bieten aufgrund ihrer unterschiedlichen Art des Übergangs auch verschiedene Möglichkeiten zur Integritätskontrolle.

- Schematische Transaktionen stellen einen funktionalen Übergang dar, ihre Wirkung ist deshalb bereits eindeutig vor ihrer Ausführung bestimmt.
- Reglementierte Transaktionen bilden einen relationalen Übergang auf der Datenbank, ihre Wirkung ist deshalb nicht eindeutig festgelegt. Dieser Umstand verursacht Schwierigkeiten, wenn bestimmte mögliche Wirkungen in einer gegebenen Situation zulässig, andere jedoch nicht zulässig sind.
- Freie Transaktionen sind in ihrer Wirkung unbestimmt, für das Kontrollsystem heißt dies, mögliche aber unzulässige Wirkungen zu verhindern.

5.5.2.1. Integritätskontrolle bei schematischen Transaktionen

Die schematischen Transaktionen sind in ihrer Wirkung eindeutig spezifiziert. Die Implementation einer schematischen Transaktion wird entsprechend zu den Aktionen (vgl. Abs. 5.4.2) im Hinblick auf die Spezifikation verifiziert.

Die Erfordernisse zur Prüfung und zu integritäts-sichernden Maßnahmen für schematische Transaktionen lassen sich bezüglich der einzelnen Integritätsarten wie folgt angeben.

- Strukturelle und semantische Datenintegrität

Die strukturellen und semantischen Integritätsbedingungen werden bei der Verifikation in die Vor- und Nachbedingung der Spezifikation mit aufgenommen, damit entfällt deren aktuelle Prüfung vor und nach der Transaktionsausführung. Dies gilt nur unter der Voraussetzung, daß die Implementation auch in bezug auf die Spezifikation als korrekt nachgewiesen wurde.

- Finale deontische Integrität

Die finale deontische Integrität ist durch die Prüfung der afinalen deontischen Integrität abgedeckt. (Die Verträglichkeit zwischen finaler und afinaler deontischer Integrität wird vorausgesetzt, sonst ist das Anwendungssystem nicht konfliktfrei.) Bei schematischen Transaktionen werden die Anforderungen aus den Normen der Miniwelt vollständig auf afinale Weise realisiert (vgl. Abs. 5.1.2).

- Ablaufintegrität

Die Gewährleistung der strukturellen Korrektheit des Ablaufs nach den Gray'schen Kriterien vereinfacht sich bei schematischen Transaktionen. Zur Sicherung der Atomarität genügt es vor Beginn der Transaktion ein Before-Image der betroffenen Datenobjekte zu nehmen, da ein lokales partielles Rücksetzen aus logischer Sicht nicht erforderlich ist. (Ach die Isolation der Transaktion wäre apriori, z.B. durch Prädikatssperren möglich (vgl. <EGLT76>).) Die Freigabe der Daten bei EOT ist für alle Transaktionsarten gleich zu behandeln.

- Übergangsintegrität

Die Vorbedingung ist am Anfang der Transaktion zu prüfen. Ist sie erfüllt kann die Transaktion ausgeführt werden. Die Wirkung der Transaktion ist durch die Nachbedingung aufgrund des gegebenen Zustands eindeutig festgelegt.

- Afinale deontische Integrität

Die Bedingungen zur afinalen deontischen Integrität sind ebenfalls zu Beginn der Transaktion zu prüfen. Ist die TRIGGER- oder PERMIT-Bedingung erfüllt, so ist der Übergang deontisch korrekt. Beim Zutreffen der REJECT-Bedingung wird die Transaktionsausführung verhindert.

Einen Überblick über die Integritätsaspekte hinsichtlich schematischer Transaktionen zum Zeitpunkt des BOT, während der Transaktionsausführung und bei EOT gibt Abb. 5-14.

Abb. 5-14.:

(* Notwendigkeit zur Integritätskontrolle bei schematischen Transaktionen *)

Zeitpunkt	BOT	< T >	EOT
Integrität			
Datenintegrität			
strukturelle Integrität	erfüllt (Definition)	verletzt	erfüllt ohne Prüfung (verifiziert)
semantische Integrität	erfüllt (Definition)	verletzt	erfüllt ohne Prüfung (verifiziert)
finale deontische Integrität	erfüllt (afinal deontisch)	erfüllt	erfüllt ohne Prüfung (verifiziert)
Operationale Integrität			
Ablaufintegrität	erfüllt (Before-Image)	erfüllt (Isolation)	erfüllt (Commitment)
Übergangsintegrität	prüfen (Vorbedingung)	prüfen (lokale Vorbed.)	erfüllt (Nachbed. verifiziert)
afinale deontische Integrität	prüfen	erfüllt	erfüllt (verifiziert)

In Abs. 5.1.1 wurde auf die Unterscheidung zwischen konstitutiven und regulativen Integritätsbedingungen hingewiesen. Bedingungen der regulativen Art sind während der Lebensdauer eines Datenbank-Anwendungssystems veränderbar. Da die regulativen durch semantische Bedingungen realisiert werden, impliziert eine Änderung regulativer Bedingungen auch eine Änderung der semantischen. Es wird dann eine Prüfung der semantischen Integritätsbedingungen auch bei EOT betroffener Transaktionen erforderlich. Die Implementationen schematischer und reglementierter Transaktionen, die bei einer Ausführung zurückgesetzt werden, sind dann neu zu verifizieren. Die Alternative dazu ist, die Implementationen aller betroffenen Transaktionen umgehend mit den neuen invarianten Bedingungen zu verifizieren.

5.5.2.2. Integritätskontrolle bei reglementierten Transaktionen

Bei reglementierten Transaktionen ist die Kontrolle der Integritätsbedingungen zu Beginn, während und am Ende der Transaktion erforderlich. Für die einzelnen Integritätsarten differieren dabei die Kontrollanforderungen und die Zeitpunkte der Überwachung.

- strukturelle und semantische Datenintegrität

Durch eine geeignete Implementation der reglementierten Transaktionen sind die Anforderungen an die strukturelle und semantische Datenintegrität soweit erfüllbar, daß eine Prüfung am Ende der Transaktion nicht erforderlich wird.

In Abs. 4.3. wurde die Struktur reglementierter Transaktionen behandelt. Die Nachbedingung dieser Transaktionsart besteht aus disjunktiv verknüpften Teilbedingungen. Die Transaktion erfüllt nach ihrer Ausführung eine dieser Teilbedingungen. Soll die reglementierte Transaktion auch die invarianten Bedingungen der Datenbank erfüllen, so ist jede der Teilbedingungen für die einzelnen Pfade konjunktiv mit den invarianten Bedingungen zu verknüpfen. Die einzelnen Pfade der Implementation sind dann zu verifizieren. Wird die in Abs. 4.3 vorgestellte Disziplin der Implementierung (die Möglichkeit eines lokalen Rücksetzens eingeschlossen) befolgt, treten keine Schwierigkeiten auf.

Die semantische und strukturelle Integrität kann jedoch bei EOT verletzt sein, wenn von der dargestellten Disziplin der Dialoggestaltung abgewichen wird.

Wird eine "freiere" Form des Dialogs erlaubt, indem benutzerinitiierte "GOTO"s zugelassen werden, können die invarianten Bedingungen verletzt werden. Der Benutzer verläßt willkürlich einen beliebigen Knoten des Dialogs (z.B. durch den direkten Aufruf einer Aktion) und setzt sich in einen anderen Knoten (damit verbunden in eine andere Dialogumgebung) der Transaktion. Diese GOTO's sind nicht verifizierbar, da sie sich erst zum Zeitpunkt der Transaktionsausführung manifestieren.

Beim Zulassen dieser benutzerinitiierten GOTO's ist die strukturelle und semantische Datenintegrität am Ende der Transaktion zu prüfen. Es ist dabei zu beachten, daß verschiedene Pfade in der Ausführung einer reglementierten Transaktion verschiedene invariante Bedingungen der Datenbank betreffen können.

- finale deontische Integrität

Bei den schematischen Transaktionen konnte die finale deontische Integrität völlig durch die Gewährleistung der afinalen deontischen Bedingungen ersetzt werden, da die Wirkung der Transaktion eindeutig bestimmt ist. Reglementierte Transaktionen können eine Menge möglicher Endzustände erzeugen. Die finale deontische Integrität hat demzufolge zu prüfen, wie ein konkret erzeugter Zustand deontisch zu bewerten ist.

Auf die Prüfung der finalen deontischen Integrität bei EOT kann verzichtet werden, wenn man eine deontische Prüfung lokal in den einzelnen Knoten des Ablaufs der Transaktion vorsieht. Hierdurch kann die deontische Zulässigkeit der Auswahl eines Pfades im Transaktionsablauf apriori geprüft werden und nicht erst bei dessen Abschluß durch EOT.

- Ablaufintegrität

Wegen der Notwendigkeit des lokalen Rücksetzens einer reglementierten Transaktion ist, im Gegensatz zur schematischen Transaktion, während des Ablaufs eine Sicherung der gemachten Änderungen erforderlich. (Es genügt die Before- und After-Image Sicherung im klassischen Sinne, Mechanismen hierzu vgl. <Reu81>.)

Die anderen Anforderungen (Isolation und Freigabe) sind entsprechend zu den schematischen Transaktionen zu behandeln.

- Übergangsintegrität

Die Vorgehensweise zur Gewährleistung der Übergangsintegrität entspricht im wesentlichen der bei schematischen Transaktionen. Eine Ausnahme besteht darin, daß sich die Vorbedingungen nicht vollständig zu Beginn der reglementierten Transaktion prüfbar sind. Ein Abbruch der Transaktion aus Gründen der Verletzung des spezifizierten Übergangs kann damit zu einem späteren Zeitpunkt noch erfolgen. Die Nachbedingung bedarf keiner Prüfung, wenn die Implementation korrekt hinsichtlich der Spezifikation ist. Läßt man die oben erwähnten benutzerinitiierten GOTO's zu, ist die spezifizierte Nachbedingung der Transaktion u.U. nicht mehr erfüllt. Es ist nicht sichergestellt, daß durch einen beliebigen Sprung zwischen zwei Eingabeknoten noch eine korrekte Implementation der Spezifikation gegeben ist. (Eine Spezifikation und Verifikation aller Möglichkeiten zum willkürlichen Wechsel eines Knotens erscheint nicht praktikabel.)

- afinale deontische Integrität

Die Prüfung der afinalen deontischen Integrität kann analog zu den Vorbedingungen des Übergangs u.U. erst während der Ausführung der Transaktion erfolgen. Dieser Sachverhalt beruht ebenfalls darauf, daß die für die Auswertung der afinalen deontischen Bedingungen benötigten Parameterwerte erst während der Ausführung bekannt werden. Dies darf jedoch nicht für die TRIGGER-Bedingung gelten. Sie muß vor der Initiierung entscheidbar sein.

Bezüglich der Prüfung der afinalen deontischen Integrität bei reglementierten Transaktionen besteht eine besondere Beziehung zur finalen deontischen Integrität. Die afinalen deontischen Bedingungen zur Initiierung in einer reglementierten Transaktion beziehen sich auf die Menge der erreichbaren Zustände. Durch die finale deontische Integrität kann diese Zustandsmenge auf einzelne Zustände in bezug auf ihre Zulässigkeit eingeschränkt sein.

Die Einschränkung läßt sich auch durch eine operationale Kontrolle erreichen, wenn man in der Implementation in den einzelnen durch die Dialogeingabe bestimmten Knoten lokale deontische Bedingungen angibt. Aufgrund dieser lokalen Initialbedingungen wird entschieden, ob in einem konkreten Zustand eine bestimmte Verzweigung zulässig oder gar geboten ist.

Die Abb 5-15 gibt einen Überblick über die Erfordernisse der Integritätskontrolle, wie sie bei der Realisierung reglementierter Transaktionen auftreten.

Abb. 5-15:

(* Notwendigkeit zur Integritätskontrolle bei reglementierten Transaktionen *)

Zeitpunkt	BOT	< T >	EOT
Integrität			
Datenintegrität			
strukturelle Integrität	erfüllt (Definition)	verletzt	erfüllt, (bei Disziplin)
semantische Integrität	erfüllt (Definition)	verletzt	erfüllt, (bei Disziplin)
finale deontische Integrität	teilweise erfüllt (afinal deontisch)	teilweise erfüllt (afinal deontisch)	teilweise erfüllt teilweise prüfen
Operationale Integrität			
Ablaufintegrität	erfüllt (Before-Image)	erfüllt (Isolation) (Before- bzw. After-Image)	erfüllt (Commitment)
Übergangsintegrität	Vorbedingung nur teilweise prüfbar	restliche Vorbed. prüfen	Nachbedingung erfüllt ohne Prüfung (verifiziert)
afinale deontische Integrität	prüfen	prüfen	erfüllt (verifiziert)

5.5.2.3. Integritätskontrolle bei freien Transaktionen

Bei freien Transaktionen ist eine Kontrolle der Integrität am Ende der Transaktion vom großen Umfang notwendig. Bestimmte Arten der Integrität wie z.B. die Übergangsintegrität entziehen sich einer Kontrolle völlig.

- strukturelle und semantische Datenintegrität

Es liegt der klassische Fall vor, daß dem Datenbanksystem nichts über die Korrektheit einer Transaktion bekannt ist. Das Kontrollsystem kann zu Beginn einer Transaktion davon ausgehen, daß die Datenbank sich in einem integeren Zustand befindet. Bei EOT einer freien Transaktion ist vom Kontrollsystem zur Laufzeit zu prüfen, ob durch die Transaktion der integere Zustand erhalten geblieben ist.

Die in den einzelnen Ebenen des Datenbank-Anwendungssystems (Datenbanksicht und Datenbank-Anwendungsmoduln) möglichen Reaktionen auf eventuelle Verletzungen der Integrität wurden in Abs. 5.3 und 5.4. bereits behandelt. Sie werden ergänzt, durch die Angabe von weiteren Triggern, wenn dem Benutzer eine operationale Sicht zur Verfügung gestellt wird (vgl. Abs. 4.4). In die Spezifikation dieser Sicht sind diese Triggermechanismen zu integrieren.

- finale deontische Integrität

Die Einhaltung von Normen läßt sich auf der Basis von freien Transaktionen nur auf der Basis finaler deontischer Integritätsbedingungen behandeln. Sie sind beim EOT der freien Transaktion zu prüfen. Abhängig vom Ergebnis der Prüfung ist gegebenenfalls eine weitere Transaktion einzuleiten oder die ausgeführte Transaktion zurückzusetzen.

- Ablaufintegrität

Es liegt im Prinzip der Fall der freien Verwendung einer Datenbanksprache vor. Es sind damit die in einem Datenbanksystem üblichen Maßnahmen zur Gewährleistung der Isolation, Atomarität und Dauerhaftigkeit durchzuführen (vgl. <Reu81>).

- Übergangs- und afinale deontische Integrität

Die Behandlung dieser beiden Integritätsarten entfällt, da sich freie Transaktionen einer diesbezüglichen Prüfung entziehen. Der von der Transaktion zu vollziehende Übergang ist nicht näher als durch die invarianten Bedingungen der Datenbank spezifiziert. Die afinale deontische Integrität einer Prüfung auf Zulässigkeit des Handelns kann nur zustandsorientiert aufgrund der erzeugten Wirkung der Transaktion vorgenommen werden.

Die Integritätskontrolle in den verschiedenen Aspekten ist für freie Transaktionen in Abb. 5-16 zusammengefaßt.

Abb. 5-16.:

(* Möglichkeiten zur Integritätskontrolle bei freien Transaktionen *)

Zeitpunkt	BOT	< T >	EOT
Integrität			
Datenintegrität			
strukturelle Integrität	erfüllt (Definition)	verletzt	prüfen
semantische Integrität	erfüllt (Definition)	verletzt	prüfen
finale deontische Integrität	unbestimmt	unbestimmt	prüfen
Operationale Integrität			
Ablaufintegrität	erfüllt (Before-Image)	erfüllt (Isolation) (Before- bzw. After-Image)	erfüllt (Commitment)
Übergangsintegrität	stets erfüllt	stets erfüllt	stets erfüllt
afinale deontische Integrität	unbestimmt	unbestimmt	prüfen (final deontisch)

5.6. Zusammenfassende Bemerkungen zur Integritätskontrolle

Die Datenintegrität umfaßt invariante Bedingungen der Datenbank die für alle Transaktionen gelten müssen. Die Kontrolle der invarianten Bedingungen erfolgt abgestuft auf den verschiedenen Ebenen des Datenbank-Anwendungssystems.

- strukturelle Datenintegrität

Die Typ- und Schlüsselintegrität kann auf der Basis elementarer Datenbank-Operationen kontrolliert werden.

Die referentielle Integrität kann erst auf der Basis der Aktionen aufrechterhalten werden, wenn man eine geeignete Disziplin bei der Erstellung der Sichten und der Aktionsprozeduren anwendet.

- semantische Datenintegrität

Die Gewährleistung der semantischen Datenintegrität ist wie die strukturelle Integrität abhängig davon, ob es sich um inter- oder intrarelationale Bedingungen handelt. Beide Arten von Bedingungen können ebenfalls bei entsprechender Disziplin auf der Ebene der Datenbank-Anwendungsmoduln gesichert werden.

- finale deontische Integrität

Die finale deontische Integrität beschreibt, abhängig von einem Ausgangszustand der Datenbank, die Zulässigkeit von Folgezuständen. Die finale deontische Integrität leitet sich aus den finalen Normen der Miniwelt ab. Die Datenbankzustände werden durch sie in vier Klassen bezüglich der deontischen Modi eingeteilt. Die Erfordernisse zur Sicherung der Integrität sind abhängig von der Beurteilung des Zustands, ob er verboten, erlaubt, geboten oder indifferent ist. Die finale deontische Integrität ist auf Transaktionsniveau zu gewährleisten.

Die operationale Integrität behandelt den transaktions-spezifischen Übergang. Die Integritätsanforderungen sind deshalb auch auf der Ebene der Transaktionen zu sichern. Bestimmte Teilaspekte der Kontrolle können aber bereits der tieferen Schicht der Datenbank-Anwendungsmoduln übertragen werden.

Die Einführung der drei Transaktionsarten freie, reglementierte und schematische Transaktionen macht deutlich, welche Anforderungen an operationaler und datenorientierter Integrität apriori beim Beginn, welche während und welche aposteri nach einer Transaktion kontrolliert werden können.

Die eingeführte afinale deontische Integrität erlaubt die Kontrolle afinaler Normen. Sie bildet aufgrund der Einteilung von Transaktionen in die vier deontischen Klassen die Möglichkeit auf die spezifischen Datenbankzustände adäquat zu reagieren.

Afinale und finale deontische Integrität werden im folgenden Kapitel subjektbezogen erweitert. Sie erlauben dann auch eine subjektbezogene "Input"-Kontrolle gegen mißbräuchliche Veränderungen der Datenbank.

6. Eine subjektbezogene Erweiterung der Integritätskontrolle

Die afinale und finale deontische Integrität, die im vorherigen Kapitel eingeführt wurde, läßt sich auf einfache Weise um einen Subjektbezug erweitern. Es wird in diesem Fall von der Betrachtung allgemein gültiger Normen auf individuell gültige Vorschriften übergegangen. Die Gründe dafür, daß eine solche Erweiterung sinnvoll erscheint, werden im folgenden dargelegt. Nach dieser Begründung wird die Struktur dieser erweiterten Bedingungen aufgezeigt und ihre Umsetzung in die in Abs. 5.3.3. und 5.5.1 angegebenen Sprachkonstrukte vorgenommen.

Der Subjektbezug ist eigentlich charakteristisch für Überlegungen im Rahmen des Datenschutzes und der Datenbank-Sicherheit. Zum Abschluß dieses Kapitels wird deshalb die subjektbezogene Integritätskontrolle in Beziehung zur Datenbank-Sicherheit gesetzt. Man erkennt, daß die subjektbezogene Integrität einen geeigneten Mechanismus zur Input-Kontrolle einer Komponente zur Gewährleistung der Datenbank-Sicherheit darstellt.

6.1. Gründe und Struktur einer subjektbezogenen Erweiterung der Integritätskontrolle

In einer Miniwelt gibt es wie in Abs. 5.1.2 und in 2.1.3 gewisse Normen und Vorschriften, die bei der Ausführung von Handlungen einzuhalten sind. Für alle Personen gültige Normen wurden im Rahmen der deontischen Integrität im vorherigen Kapitel bereits behandelt. In der Miniwelt dürfen aber bestimmte Personen nur bestimmte Handlungen ausführen bzw. sie sind sogar in bestimmten Situationen zu deren Ausführung verpflichtet.

In der Miniwelt existiert ein "Rechtssystem", das diese Rechte und Pflichten des Handelns einzelner Personen (-gruppen) festlegt. In einer betrieblichen Miniwelt, wie z.B. der des Pharmazeutischen Außendienstes, existieren Betriebsvereinbarungen, die das Handeln der Außendienstmitarbeiter regeln. Die Pflichten und Rechte von Mitarbeitern eines Unternehmens sind in den Arbeitsanweisungen und -richtlinien festgelegt. Die in der Miniwelt beschäftigten Personen haben beim Arbeiten mit dem Datenbank-Anwendungssystem, das durch seine Transaktionen einen Teil der anfallenden Handlungen realisiert, diese betrieblichen Vereinbarungen einzuhalten.

Jedem Mitarbeiter einer Unternehmung ist eine Tätigkeitsbeschreibung zugeordnet, in der seine Aufgaben und Pflichten, die er in der betrieblichen Miniwelt zu erfüllen hat, festgelegt sind. Die Tätigkeitsbeschreibungen schaffen die Voraussetzungen für das Handeln der Personen innerhalb eines Betriebes. Es sind primär nur Handlungen durch eine Person zulässig, wenn sie durch die Tätigkeitsbeschreibung abgedeckt ist. Die Betriebsinteressen- oder auch Unternehmensziele ergänzen die Betriebsvereinbarungen. Sie legen die Ziele fest, an denen die Handlungen in der Miniwelt auszurichten sind.

Aufbau von Maximen und Vorschriften

Die allgemeine Rechtfertigung von Handlungen aufgrund der in einer Miniwelt geltenden Normen wurde bereits behandelt (vgl. Abs. 2.1.3). Eine Handlung gilt im allgemeinen Sinne als gerechtfertigt, wenn ihre Aktualisierung auf eine geltende Norm zurückführbar ist. Die Normen einer Miniwelt bildeten die Begründung für die Einführung der (afinalen und finalen) deontischen Integrität (vgl. Abs. 5.1.2).

In der Wissenschaftstheorie (z.B. <Lor79>, S.121) wird eine Handlung durch die Maximen erklärt, die eine Person verfolgt. Maximen sind im Gegensatz zu Normen individuelle, personenbezogene Aufforderungen zur Verfolgung eines Zwecks. Kulturwissenschaftlich betrachtet entstehen Normen durch Maximengenese (vgl. <Sch76>, S.199ff). Vorschriften legen fest, welche Personen zur Befolgung welcher Maximen verpflichtet sind. Im Rahmen einer Gemeinschaft gibt es entsprechende Zwangsmittel (vgl. <Lor79>), um die Einhaltung der Vorschriften zu gewährleisten. In einem Datenbank-Anwendungssystem hat dies durch eine Erweiterung der Integritätskontrolle zu geschehen.

Die Subjekte

In die Vorschriften zur deontischen Bewertung eines Übergangs geht deshalb die Menge der Personen der Miniwelt mit ein. Die Angabe der Personen der Miniwelt kann explizit durch Aufzählung oder durch die Angabe von Eigenschaften erfolgen. In einer betrieblichen Organisation wird man die Eigenschaften (man spricht auch von Rollen <Cha78>) zur Festlegung von Subjektklassen heranziehen. Die Subjektklassen werden charakterisiert durch die innerbetriebliche Organisation (oder innerbetriebliche "Funktion"). Eine konkrete Person erfüllt in der betrieblichen Miniwelt eine "Rolle", die ihr zugeordnet wurde. Die Möglichkeiten ihres Handels ist durch ihre Rolle bestimmt.

Am Beispiel der Pharma-Datenbank wird dies insofern deutlich, daß die zulässigen Operationen eines Mitarbeiters abhängig sind von seiner Rolle z.B. Gruppenleiter des Gebiets "Franken". Wird die Person versetzt, verliert es die mit der Rolle verbundenen Rechte und Pflichten *).

Für Überwachung der subjektbezogenen Integrität sind diese Rollen von Interesse. Mittels einer Abbildungsfunktion sind den konkreten Personen der Miniwelt ihre Rollen zuzuordnen oder bei gebotenen schematischen Transaktionen übernimmt das Datenbank-Anwendungssystem eine dieser Rollen. Alle Rechte, Pflichten und Verbote in der Form subjektbezogener Integritätsbedingungen sind an diese Rollen geknüpft.

Bei der Initiierung einer Transaktion durch eine Person der Miniwelt ist der aus den Sicherheitsüberlegungen bekannte Identifikations- und Authentifikationsvorgang (<FSW81>, S.247) durchzuführen. Die Person hat sich zu identifizieren, z.B. durch die Eingabe einer Benutzerkennung. Danach erfolgt eine Authentifikation, in der die Authentizität der Person geprüft wird. Die gängige Methode hierfür ist die Verwendung von Paßwörtern. Eine einfache Erweiterung und für kommerzielle Systeme praktikabel (vgl. <Cha78>) ist die Einbeziehung der Zeit und des Terminals über das die Anforderung eingeht. Man erhält für die Zordnung einer Person zur Rolle folgende Funktionen (vgl. hierzu auch <Cha79>, S.41):

auth1: PERSON X PASSWORD X TERMINAL X TIME ----> SUBJEKT

auth2: SUBJECT X DATABASE ----> ROLE

*) Bemerkung: (Die Überwachung der Vorschriften einer konkreten Miniwelt kann von einer kleinlichen Gängelung des Handelns der Personen bis zur Ausweitung der gewährten persönlichen Freiräume führen. Entscheidend ist die "vernünftige" Gestaltung der Vorschriften. Sie ist nicht Sache des Datenbank-Anwendungssystems, sondern der Verantwortlichen.)

Die Subjekte im Datenbank-Anwendungssystem sind als abstrakte Subjekte zu verstehen. Durch die Verwendung abstrakter Subjekte wird es möglich Klassen von Personen zu einem abstrakten Subjekt zusammenzufassen. Die abstrakten Subjekte ihrerseits können wiederum zu abstrakten Subjekten zusammengefaßt werden. Den Subjekten wird aufgrund von Eigenschaften, die in der Datenbank abgelegt sind, eine Rolle zugewiesen. Die Zugehörigkeit einer Person zu einer Rolle ist deshalb in einem Datenbanksystem einfach zu handhaben.

Diese Indirektion in der Zuordnung bietet eine gute Möglichkeit organisatorische Maßnahmen, wie das Versetzen von Personen in einer Unternehmung unabhängig von den in den Integritätsbedingungen festgelegten Anforderungen vorzunehmen.

Struktur subjektbezogener Integritätsbedingungen

Die Strukturen der Vorschriften entsprechen denen der Normen, erweitert um die Subjekte, die von der Vorschrift betroffen sind. Im Datenbank-Anwendungssystem sind die Anforderungen an die deontische Integrität lediglich zu individualisieren. Aus dem Abs. 5.1.2 ist bekannt, daß Übergänge in der Datenbank nach den deontischen Modi (geboten, verboten, erlaubt, indifferent) zu bewerten sind. Die Bewertung des Übergangs wird jetzt abhängig vom Individuum, das die entsprechende Transaktion initiiert oder ausführt.

Die Struktur der Vorschriften ergibt in der Struktur eine subjektbezogene finale deontischen Integritätsanforderung der folgenden Form (analog zu Abs. 5.3.3):

```
si-int-i : DATABASE-OLD X DATABASE-NEW X ROLE  
-----> {deont. Modi}
```

Damit ist die allgemeine Struktur für die Übertragung der finalen Form der Vorschriften gegeben.

Die Vorschriften können auch afinal angegeben werden, es ergibt sich dann eine deontische Bedingung der Form:

sd-int-i : DATABASE-OLD X T X INPUT X ROLE

-----> { deont. Modi }

wobei DATABASE-OLD die Menge der möglichen Ausgangszustände und DATABASE-NEW die Menge der Enzustände der Datenbank beschreibt, T eine Transaktion angibt und ROLE für die Menge der abstrakten Subjekte (mit einsprechenden Eigenschaften) des Anwendungssystems steht.

Analog zur afinalen deontischen Integrität (Abs. 5.1.2) sind die möglichen Übergänge in vier Äquivalenzklassen bezüglich der Faserung über die deontischen Modi der in einer Anwendung geltenden Vorschriften einzuteilen. Der wesentliche Unterschied der Vorschriften zu den Normen durch den Subjektbezug wird deutlich. Aus der deontischen Bewertung der Übergänge ergeben sich für die Subjekte einer Miniwelt Rechte, Pflichten und Verbote. Die subjektbezogenen Rechte ergeben sich aus der Äquivalenzklasse der erlaubten Übergänge beschränkt auf ein Subjekt. Entsprechend ergeben sich die Pflichten eines Subjekts aus der Klasse der gebotenen Übergänge.

Die Vorschriften stellen sich somit für das Datenbank-Anwendungssystem als subjektbezogene Integritätsbedingungen dar. Zur Beurteilung einer konkreten Transaktion sind die deontischen Äquivalenzklassen bezüglich der Vorschriften zu bilden. Die funktionale Spezifikation der Transaktion ist um eine Klasse von Rollen "ROLE" zu erweitern. Die Äquivalenzklassen bezüglich der Bewertung von Übergängen ergeben sich dann mit

$db-old \in DATABASE-OLD$, $db-new \in DATABASE-NEW$, $r \in ROLE$
und $T(db-old, input, r) = (db-new, output)$ als

$$S\text{-INT(mod)} = \{ T \quad \forall \text{ si-int-i} \\
\quad (\text{si-int-i}(\text{db-old}, \text{db-new}, r) = \text{mod}) \\
\quad \vee \quad \forall \text{ sd-int-i} \\
\quad (\text{sd-int}(\text{db-old}, \text{input}, T, r) = \text{mod}) \}$$

wobei $\text{mod} \in \{ \text{deont. Modi} \}$.

Die Beurteilung einer konkret ausgeführten Transaktion bezüglich ihrer Zulässigkeit erfolgt analog zur Integrität über die deontische Klassenzugehörigkeit.

6.2. Subjektbezogene deontische Integritätsbedingungen

Die Sprachkonstrukte zur der Spezifikation von Transaktionen und zur Formulierung der finalen deontischen Bedingungen im Konzeptionellen Schema sind um Möglichkeiten zur Angabe des Subjektbezugs zu erweitern. Ansonsten ergeben sich keine weiteren Unterschiede in der Darstellung der Integritätsbedingungen.

Subjektbezogene afinale deontische Bedingungen

Die Spezifikation der Transaktion wurde (vgl. Abs. 5.5.1) zur Gewährleistung der afinalen deontischen Integrität um die sogenannten Initialbedingungen erweitert. Die Initialbedingungen bestehen aus drei Teilen. Die einzelnen Teile geben jeweils die Kriterien für die deontische Beurteilung der Transaktionsausführung in einem konkret vorliegenden Zustand der Datenbank an. Bezüglich dieses Zustands ist die Transaktion in eine der deontischen Äquivalenzklassen einzuordnen und das Kontrollsystem kann dann darauf spezifisch reagieren.

Die Ausführung einer Transaktion ist demnach geboten, wenn in den Initialbedingungen die TIGGER-Bedingung erfüllt ist.

Die TRIGGER-Bedingung läßt sich einfach erweitern:

INIT-CONDITION:

TRIGGER FOR <roles> ON <predicate>;

wobei <roles> eine Menge von abstrakten Subjekten des Datenbank-Anwendungssystems spezifiziert. (In den Beispielen in Anhang 3 und 4 erfolgt die Spezifikation der Subjekte in ihren Rollen aufgrund von Eigenschaften, die in der Datenbank abgelegt sind.)

Die anderen Klauseln der Initialbedingungen ergeben sich analog,

für die explizite Erlaubnis der Transaktion:

PERMIT FOR <roles> ON <predicate>;

für die Verbote:

REJECT FOR <roles> ON <predicate>;

Die Bedeutung der subjektbezogenen Initialbedingungen ergibt sich durch die Angabe mehrerer Bedingungen eines Typs. Es wird damit möglich gleiche Transaktionen unter verschiedenen Voraussetzungen für verschiedene Benutzer zu erlauben *).

Als Beispiel, die Spesensätze werden für einem Pharma-refernten und seinen Gruppenleiter differieren. Die REJECT-Klauseln bezüglich der beiden Rollen werden sich folglich unterscheiden.

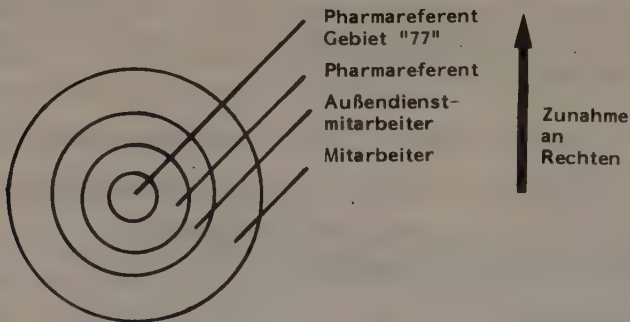
Aus diesem Beispiel erkennt man auch, daß es in einer Anwendung sinnvoll sein kann, wenn man die Rollen wei-

*) In den Initialbedingungen kann vom direkten Eintrag einer Rolle abgewichen werden. Durch die Anwendung eines "Schlüssel-Schloß-Mechanismus" kann eine weitere Indirektion bei der Aktivierung von Transaktionen erreicht werden.

tergehend strukturiert, z.B. die einzelnen Pharmareferenten wieder zusammenfaßt etc.

Abb. 6-1.:

(* Struktur von Rollen *)



Subjektbezogene finale deontische Bedingungen

Völlig analog zur Erweiterung der afinalen deontischen Bedingungen in den Transaktionen kann man die Erweiterung in den finalen deontischen Bedingungen des Konzeptionellen Schemas der Datenbank vornehmen.

Man erhält für einen expliziten erlaubten Übergang folgende Erweiterung:

```

DEFINE PERMISSION
FOR <roles> TRANSITION
FROM
  (<target-list> WHERE <qualification>)
TO
  (<target-list> WHERE <qualification>)
  
```

Gebotene Übergänge der Datenbank ergeben sich als:

DEFINE OBLIGATION

FOR <roles> TRANSITION

FROM

(<target-list> WHERE <qualification>)

TO

(<target-list> WHERE <qualification>)

Die subjektbezogenen Verbote:

DEFINE PROHIBITION

FOR <roles> TRANSITION

FROM

(<target-list> WHERE <qualification>)

TO

(<target-list> WHERE <qualification>)

Die finalen deontischen Bedingungen lassen sich entsprechend den Aussagen in Abs. 5.3.3. mit Triggern versehen. Man kann damit erreichen, daß auf bestimmte Verletzungen des Datenbankzustands subjektspezifische Transaktionen angestoßen werden. Aus der Pharmadatenbank als Beispiel, der Pharmareferent wird beim Einloggen in des System aufgefordert, die Transaktion Tagesbericht zu erstellen, bevor er andere Transaktionen z.B. Besuchsplanung etc. initiieren kann.

6.3. Subjektbezogene Integrität ein Beitrag zur Sicherheitskontrolle

Durch die subjektbezogene Integrität wird bezüglich der Sicherheit der Datenbank eine Inputkontrolle durchgeführt. Verletzungen der Datenbank durch nicht erlaubte Eingaben von Benutzern kann ausgeschlossen werden. Die Erkennung einer Verletzung und die eventuelle Verhinderung hängt von der Prüfbarkeit der Bedingungen ab, die in Abs. 5.5.2 behandelt wurde.

Die subjektbezogenen Integritätsbedingungen ermöglichen eine "Inputkontrolle" für Datenbank-Anwendungssysteme. Es wird eine Verletzung des Zustands der Datenbank ausgeschlossen. Die subjektbezogene Integritätskontrolle ist deshalb auch Bestandteil eines Kontrollsystems zur Gewährleistung der Sicherheit in einem Datenbank-Anwendungssystem. Die subjektbezogenen Integritätsbedingungen könnte man deshalb auch als Sicherheitsbedingungen zur Kontrolle der subjektspezifischen Dateneingabe verstehen.

Die subjektbezogene Integritätskontrolle verhindert die mißbräuchliche und irrtümliche Erzeugung eines Datenbankzustands durch einen Benutzer. Ein Benutzer ist nur zur Erzeugung bestimmter Zustände befugt. Erzeugt er andere Zustände kann das Kontrollsystem eingreifen und Maßnahmen zur Aufrechterhaltung der Datenbank-Sicherheit einleiten.

Ein Datenbank-Anwendungssystem benötigt neben dieser Inputkontrolle, die eine Verfälschung der Daten verhindert, eine leistungsfähige "Outputkontrolle", wie sie von Jan Schlörer in «Schl82» erörtert wird. Die Outputkontrolle überwacht Verletzungen der Miniwelt bezüglich eines Individuums, die durch eine Transaktionsausführung entstehen können.

Input- und Outputkontrolle ergänzen sich in der Gewährleistung der Datenbank-Sicherheit zu einem umfassenden Sicherheitssystem für Datenbank-Anwendungen. Die Anforderungen aus dem Bundesdatenschutzgesetzes <BDSG>, S.27 an Datenbank-Anwendungssysteme sind damit erfüllbar.

7. Zusammenfassung

In der Wedekind'schen Gesamtmethodologie zur Entwicklung von Datenbank-Anwendungssystemen (vgl. <Wed79a>) werden sechs Zustände (Aufgabenstellung (Z1), Objekttypenkonstruktion (Z2), Erstellung des Konzeptionelles Schema (Z3), Erstellung der Externen Schemata (Z4), Entwurf der Transaktionen (Z5), Lösung der gestellten Aufgabe (Z6)) in drei Ebenen (fachliche Ebene (Z1,Z6), konstruktive Ebene (Z2,Z4), Schemaebene (Z3,Z6)) angegeben, die bei der Erstellung einer Datenbank-Anwendungssystems zu durchlaufen sind.

Die vorliegende Arbeit befaßte sich mit den Methoden und Strukturen eines Datenbank-Anwendungssystems, die in den Zuständen Z4 und Z5 erforderlich sind.

Der strukturelle Aufbau eines Datenbank-Anwendungssystems wurde aus der allgemeinen Handlungstheorie begründet und auf der Basis des Transaktionskonzepts dargestellt (Kapitel 2). Zur Formulierung des Datenflusses zwischen Datenbank und Transaktionen wurde das Sichtenkonzept der Datenbanken erweitert. Es wurde dabei aufgezeigt, wie sich das Konzept der Abstrakten Datentypen in Form Virtueller Abstrakter Datentypen in die Architektur von Datenbank-Anwendungssystemen einbringen (Kapitel 3) läßt. Die Objekte der Abstrakten Datentypen werden in der Datenbank realisiert und über Externe Schemata zur Verfügung gestellt. Auf den Externen Schemata sind die den Typ charakterisierenden Operationen zu definieren. Durch diese Vorgehensweise entsteht eine operationale Sicht der Datenbank bestehend aus einer Menge von Virtuellen Abstrakten Datentypen.

Eine Klassifikation der Transaktionen nach dem Grade ihrer Automatisierbarkeit in freie, reglementierte und schematische Transaktionen wurde vorgenommen. Die Auswirkungungen der strukturellen Unterschiede für den Aufbau eines Datenbank-Anwendungssystems wurden behandelt (Kapitel 4).

Schematische Transaktionen stellen einen funktionalen Übergang von einem Datenbankzustand in einen anderen dar. Reglementierte Transaktionen erlauben dem Anwender während der Transaktionsausführung aus verschiedenen Alternativen eine Zustandstransformation nach seinem subjektiven Willen auszuwählen. Freie Transaktionen entsprechen kreativem Handeln, ihre Wirkung ist nicht vorher bestimmt; sie unterliegen nur den allgemeinen Integritätsanforderungen bezüglich der Datenbank.

Die Einteilung der Transaktionen ermöglichte auch ein abgestuftes Konzept zur Integritätskontrolle zu entwickeln. Hierzu wurden die gängigen Integritätsanforderungen klassifiziert und zu den Transaktionsarten in Beziehung gesetzt (Kapitel 5). Durch die Betrachtung der Handlungen der Miniwelt in ihrer Realisierung als Transaktionen konnte die Integritätskontrolle erweitert werden. Die eingeführte finale und afinale deontische Integrität, die beide auf der Normenlogik basieren, ermöglichen eine differenzierte Behandlung der Anforderungen aus der Miniwelt. Die finale Integrität ermöglicht die Bewertung der erzeugter Datenbankzustände anhand der vier deontischen Klassen (erlaubt, geboten, verboten und indifferent). Die afinale deontische Integrität ermöglicht eine Kontrolle der Zulässigkeit von Transaktionen hinsichtlich ihrer Wirkung auf die Datenbank. Abhängig von der deontischen Beurteilung eines Übergangs wird vom Kontrollsystem reagiert, d.h. es werden weitere (Trans-)Aktionen angestoßen (Kapitel 5).

Diese erweiterte deontische Integritätskontrolle ermöglicht auch eine personenbezogene "Inputkontrolle" der in die Datenbank eingegebenen Daten. Die neben der "Outputkontrolle" einen wesentlichen Beitrag zur Sicherheit eines Datenbank-Anwendungssystem leisten kann (Kapitel 6).

B. Literatur

- <Adi81> Adiba, M.:
Derived Relations: A Unified Mechanism for Views, Snapshots
and Distributed Data,
Proc. 7th Conf. on VLDB, Cannes, 1981, S.293-305.
- <ALM82> Allen, F.W.; Loomis, M.E.S.; Mannino, M.V.:
The Integrated Dictionary/Directory System,
ACM Comp. Surveys, Vol.14 (1982), No.2, S.245-275.
- <Ast76> Astrahan, M.M.; et al.:
System R: a Relational Approach to Database Management,
ACM TODS, Vol.1 (1976), No.2, S.97-137.
- <Ast80> Astrahan, M.M.; et al.:
A History and Evaluation of System R,
IBM Research Report, RJ 2843, San Jose, 1980.
- <Atk78> Atkinson, M.P.:
Programming Languages and Databases,
Proc. 4th Conf. on VLDB, West Berlin, 1978, S.408-419.
- <BS81> Bancelhon, F.; Spyrtos, N.:
Data Base Mappings,
INRIA Rapports de Recherche, No. 62/63, Le Chesnay, 1981.
- <BS81a> Bancelhon, F.; Spyrtos, N.:
Update Semantics of Relational Views,
ACM TODS, Vol.6 (1981), No.4, S.557-575.
- <BDW81> Baroody, A.J.; DeWitt, D.J.:
An Object-Oriented Approach to Database System Implementation,
ACM TODS, Vol.6 (1981), No.4, S.576-601.
- <BHR80> Bayer, R.; Heller, H.; Reiser, A.:
Parallelism and Recovery in Database Systems,
ACM TODS, Vol.5 (1980), No.2, S.139-156.
- <BR81> Best, E.; Randell, B.:
A Formal Model of Atomicity in Asynchronous Systems,
Acta Informatica, Vol.16 (1981), S.93-124.
- <Bla79> Blasgen, M.W.; et al.:
System R: An Architectural Update,
IBM Research Report, RJ 2581, San Jose, 1979.
- <Bro80> Brodie, M.L.:
The Application of Data Types to Database Semantic Integrity,
Information Systems, Vol.5 (1980), S.287-296.

- <Bro81> Brodie, M.L.:
On Modelling Behavioural Semantics of Databases,
Proc. 7th Conf. on VLDB, Cannes, 1981, S.32-42.
- <BZ81> Brodie, M.L.; Zilles, S.N. (Hrsg.):
Proc. of the Workshop on Data Abstraction, Databases, and
Conceptual Modelling,
Pingree Park, SIGPLAN Notices, Vol.16 (1981) No.1.
- <Bro82> Brown, J.W.:
Controlling the Complexity of Menu Networks,
CACM, Vol.25 (1982), No.7, S.412-418.
- <BDSG80> Bundesminister des Innern (Hrsg.):
Betrifft: Bundesdatenschutzgesetz, Bonn, 1980.
- <CGT75> Chamberlin, D.D.; Gray, J.N.; Traiger, I.L.:
Views, Authorization and Locking in a Relational Data Base
System, Proc. National Computer Conference, 1975,
AFIPS Press, Montvale, Vol.44, S.425-430.
- <Cha76> Chamberlin, D.D.; et al.:
SEQUEL 2: A Unified Approach to Data Definition, Manipulation,
and Control, IBM Research Report, RJ 1798, San Jose, 1976.
- <Cha78> Chamberlin, D.D.; et al.:
Data Base System Authorization,
DeMillo, R.A. et al.(Hrsg.): Foundations of Secure Computation,
Academic Press, New York, 1978, S.39-52.
- <Cha80> Chamberlin, D.D.; et al.:
Support for Repetitive Transactions and Ad-Hoc Queries in
System R, IBM Research Report, RJ 2794, San Jose, 1980.
- <Cha80> Chamberlin, D.D.:
A Summary of User Experience with SQL Data Sublanguage,
IBM Research Report, RJ 2767, San Jose, 1980.
- <Che76> Chen, P.P.-S.:
The Entity-Relationship Model - Toward a Unified View of Data,
ACM TODS, Vol.1 (1976), No.1, S.9-36.
- <Cod70> Codd, E.F.:
A Relational Model for Large Shared Data Banks,
CACM, Vol.13 (1970), No.6, S.377-387.
- <Cod71> Codd, E.F.:
Further Normalization of the Data Base Relational Model,
Rustin R. (ed.): Data Base Systems, New York, 1971, S.33-64.

- <Cod79> Codd, E.F.:
Extending the Relational Model,
ACM TODS, Vol.4 (1979), No.4, S.397-434.
- <Cod82> Codd, E.F.:
Relational Database: A Practical Foundation for Productivity,
CACM, Vol.25 (1982), No.2, S.109-117.
- <Dat81> Date, C.J.:
An Introduction to Database Systems (3.Aufl.)
Addison - Wesley, Reading, 1981.
- <Dat81a> Date, C.J.:
Referential Integrity,
Proc. 7th Conf. on VLDB, Cannes, 1981, S.2-12.
- <Dat82> Date, C.J.:
A Formal Definition of the Relational Model,
ACM SIGMOD Record, Vol.13 (1982), No.1, S.18-29.
- <Dav80> Davidson, D.:
Essays on Action and Events, Clarendon Press, Oxford, 1980.
- <Dav72> Davies, C.T.:
A Recovery/Integrity Architecture for Data System,
IBM Technical Report, TR 02.528, San Jose, 1972.
- <Dav78> Davies, C.T.:
Data Processing Spheres of Control,
IBM System Journal, Vol.17 (1978), No.2, S.179-198.
- <DB78> Dayal, U.; Bernstein, P.A.:
On the Updatability of Relational Views,
Proc. 4th Conf. on VLDB, West Berlin, 1978, S.368-377.
- <DB82> Dayal, U.; Bernstein, P.A.:
On the Correct Translation of Update Operations on Relational
Views, ACM TODS, Vol.7 (1982), No.3, S.381-416.
- <EKW78> Ehrig, H.; Kreowski, H.-J.; Weber, H.:
Algebraic Specification Schemas for Data Base Systems,
Proc. 4th Conf. on VLDB, West Berlin, 1978, S.427-440.
- <EC75> Eswaran, K.P.; Chamberlin, D.D.:
Functional Specification of a Subsystem for Data Base Integrity,
IBM Research Report, RJ 1601, San Jose, 1975.
- <Esw76> Eswaran, K.P.:
Specifications, Implementations and Interactions of a Trigger
Subsystem in an Integrated Database System,
IBM Research Report, RJ 1820, San Jose, 1976.

- <EGLT76> Eswaran, K.P.; Gray, J.N.; Lorie, R.A.; Traiger, I.L.:
On the Notions of Consistency and Predicate Locks in a Data
Base System, CACM, Vol.19 (1976), No.11, S.624-633.
- <Fag78> Fagin, R.:
On an Authorization Mechanism,
ACM TODS, Vol.3 (1978), No.3, S.310-319.
- <FSW81> Fernandez, E.B.; Summers, R.C.; Wood, C.:
Database Security and Integrity,
Addison - Wesley Publ. Comp., Reading, 1981.
- <Fie83> Fiedler, A.:
Ein Subsystem zur Versionenkontrolle in relationalen Datenbanken
Studienarbeit am IMMD(VI), (in Bearbeitung)
Universität Erlangen, 1983.
- <FR78> Foucaut, O.; Rolland, C.:
Concepts for Design of an Information System -
Conceptual Schema and its Utilization in the REMORA Project,
Proc. 4th Conf. on VLDB, West Berlin, 1978, S.342-350.
- <GM82> Gardarin, G.; Melkanoff, M.:
Concurrency Control Principles in Distributed and Centralized
Databases,
INRIA Rapports de Recherche, No. 113, Le Chesnay, 1982.
- <GEI82> GEI - Gesellschaft f. Elektronische Informationsverarbeitung
m.b.H. (Hrsg.): IDM - Intelligent Database Machine,
Systembeschreibung, Manual IDM-SY-01, 06/82, Aachen, 1982.
- <Gol70> Goldman, A.I.:
A Theory of Human Action,
Prentice - Hall, Englewood Cliffs, N.J., 1970.
- <Gol81> Goldstein, B.S.:
Constraints on Null Values in Relational Databases,
Proc. 7th Conf. on VLDB, Cannes, 1981, S.101-110.
- <Gra77> Gray, J.N.:
Notes on Data Base Operating Systems,
"Operating Systems - An Advanced Course",
Lecture Notes in Computer Science, Vol.60, Springer Verlag,
Heidelberg, 1976, S.394-481.
- <Gra80> Gray, J.N.:
A Transaction Model,
de Bakker, J.W.; Van Leeuwen, J. (Hrsg.):
Proc. Seventh Colloquium Automata, Languages and Programming,
Springer Verlag, Heidelberg, 1980, S.282-298.

- <Gra81> Gray, J.N.:
The Transaction Concept: Virtues and Limitations,
Proc. 7th Conf. on VLDB, Cannes, 1981, S.144-154.
- <Gra81a> Gray, J.N.:
An Approach to End-User Application Design,
TANDEM Technical Report TR 81.1,
TANDEM Computers Inc., Cupertino, 1981.
- <Gri82> Gries, D.:
The Science of Programming,
Springer Verlag, Heidelberg, 1981.
- <Gro82> Grosch, J.:
Eine Programmiersprache mit mengentheoretischen Konstrukten
und deren effiziente Implementierung,
Arbeitsberichte des IMMD, Universität Erlangen,
Band 15 (1982) Nr.9.
- <Här78> Härder, Th.:
Implementierung von Datenbanksystemen,
Carl Hanser Verlag, München, 1978.
- <HR83> Härder, Th.; Reuter, A.:
Concepts for Implementing a Centralized Database Management
System, Schneider, H.J. (Hrsg.):
Proc. of ICS on Application Systems Development,
Nürnberg, B.G. Teubner, Stuttgart, 1983, S. 28-59.
- <HML78> Hammer, M.; McLeod, D.:
The Semantic Data Model: A Modelling Mechanism for Database
Applications, Proc. Intern. Conf. on Management of Data,
Austin, 1978, S.26-35.
- <HB80> Hammer, M.; Berkowitz, B.:
DIAL: A Programming Language for Data Intensive Applications,
Proc. Intern. Conf. on Management of Data, Santa Monica, 1980,
S.75-92.
- <IBM81> IBM (Hrsg.):
SQL / Data System Concepts and Facilities,
Progr.Nr. 5748-XXJ, Manual GH 24-5013-0.
- <ISO82> ISO TC97/SC5/WG 3:
Concepts and Terminology for the Conceptual Schema and the
Information Base, Griethuysen J.J.v. (Hrsg.): ISO - Report,
ISO/TC97/SC5-N695, 1982.
- <Jef76> Jefferson, D.K.:
The Role of the External Schema,
Jardine D.A. (Hrsg.): Proc. of 2nd SHARE Conf. on DBMS,
Montreal, 1976, S.67-80.

- <Jon78> Jones, A.K.:
The Object Model: A Conceptual Tool for Structuring Software,
"Operating Systems - An Advanced Course",
Lecture Notes in Computer Science, Vol.60, Springer Verlag,
Heidelberg, 1976, S.8-16.
- <Kal72> Kalinowski, G.:
Einführung in die Normenlogik,
Athenäum - Verlag, Frankfurt, 1972.
- <KS82> Kaiser, P.; Stetina, I.:
A Dialog Generator,
Software - Practice and Experience, Vol.12 (1982), S.693-707.
- <Kam80> Kambartel, F.:
Pragmatische Grundlagen der Semantik,
Gethmann, C.F. (Hrsg.): Theorie des wissenschaftlichen
Argumentierens, Suhrkamp Verlag, Frankfurt, 1980, S.95-113.
- <KL73> Kamlah, W.; Lorenzen, P.:
Logische Propädeutik, Vorschule des vernünftigen Redens,
(2. verbesserte Auflage), B.I.-Hochschultaschenbücher,
Band 227, Mannheim, 1973.
- <Kel81> Keller, A.M.:
Update to Relational Databases Through Views Involving Joins,
IBM Research Report, RJ 3282, San Jose, 1981.
- <Ker82> Keramidis, S.:
Eine Methode zur Spezifikation und korrekten Implementierung
von asynchronen Systemen,
Arbeitsberichte des IMMD, Universität Erlangen,
Band 15 (1982) Nr.4.
- <Lea81> Leavenworth, B.:
Database Views Using Data Abstraction,
IBM Research Report, RC 8722, Yorktown Heights, 1981.
- <Len78> Lenk, H.:
Handlung als Interpretationskonstrukt,
Lenk, H. (Hrsg.): Handlungstheorien - interdisziplinär,
Band 2, Wilhelm Fink Verlag, München, 1978, S.279-350.
- <Len80> Lenk, H.:
Graphen- und Verbandstrukturen in formalen Handlungstheorien,
Lenk, H. (Hrsg.): Handlungstheorien - interdisziplinär,
Band 1, Wilhelm Fink Verlag, München, 1980, S.137-165.
- <Lin79> Lindsay, B.G.; et al.:
Notes on Distributed Databases,
IBM Research Report, RJ 2571, San Jose, 1979.

- <LS75> Lorenzen, P.; Schwemmer, O.:
Konstruktive Logik, Ethik und Wissenschaftstheorie,
B.I.-Hochschultaschenbücher, Band 700, Mannheim, 1975.
- <Lor79> Lorenzen, P.:
Wissenschaftstheorie und Nelsons Erkenntnistheorie am Beispiel
Geometrie und Ethik, RATIO, Band 21 (1979) Heft 2, S.109-123.
- <McL76> McLeod, D.:
High Level Domain Specification in a Relational Data Base
System, Proc. of ACM SIGPLAN/SIGMOD Conf. on "Data:
Abstraction, Definition and Structure", Salt Lake City,
SIGPLAN Notices, Vol. II, Special Issue, 1976, S.47-57.
- <McL78> McLeod, D.:
A Semantic Data Base Model and Its Associated Structured User
Interface, Ph.D. - Thesis, MIT, Cambridge, 1978.
- <McL79> McLeod, D.:
A Database Transaction Specification Methodology for End-Users,
Technical Report TR.No. 79-6, Computer Science Department,
University of Southern California, Los Angeles, 1979.
- <Mel81> Melkanoff, M. (Chairman):
Inoffical Panel Session on Behavioral Modelling,
7th Intern. Conf. on VLDB, Cannes, 1981.
- <Min76> Minsky, N.:
Intentional Resolution of Privacy Protection in Database
Systems, CACM, Vol.19 (1976), No.3, S.148-159.
- <MS83> Müller, Th.; Steinbauer, D.:
Eine Sprachschnittstelle zur Versionenkontrolle in
CAM-Datenbanken,
erscheint in: GI - 13.Jahrestagung, Informatik Fachberichte,
Springer Verlag, Heidelberg, 1983.
- <Pao78> Paolini, P.:
An Alternative Structure for Data Base Management Systems,
Proc. 4th Conf. on VLDB, West Berlin, 1978, S.243-254.
- <PP77> Paolini, P.; Pelagatti, G.:
Formal definition of Mapping in a Data Base,
Proc. Intern. Conf. on Management of Data, 1977, S.40-46.
- <Pir82> Pirotte, A.:
A Precise Definition of Basic Relational Notions and of the
Relational Algebra,
ACM SIGMOD Record, Vol.13 (1982), No.1, S.30-45.

- <RRUZ82> Rebsamen, J.; Reimer, M.; Ursprung, P.; Zehnder, C.A.:
LIDAS - A Database System for the Personal Computer Liliith
- The Database Management,
Berichte des Instituts für Informatik Nr.50, ETH-Zuerich, 1982.
- <RZ82> Rebsamen, J.; Zehnder, C.A.:
Automatische Erzeugung von konsitzenzerhaltenden Transaktionen:
Ein Hilfsmittel zur Datenmanipulation auf Arbeitsplatzrechnern
GI - 12.Jahrestagung, Informatik-Fachberichte, Nr. 33,
Springer Verlag, Heidelberg, 1982, S.595-606.
- <RS81> Reimer, M.; Schmidt, J.W.:
Transaction Procedures with Relational Parameters,
Berichte des Instituts für Informatik Nr.45, ETH-Zuerich, 1981.
- <RT82> Relational Technology (Hrsg.):
INGRES Reference Manual, (Version 1.4., July 1982)
Relational Technology Inc., Berkley.
- <RT82a> Relational Technology (Hrsg.):
INGRES Equel (Embedded Query Language), (Version 1.4., 1982)
Relational Technology Inc., Berkley.
- <Res67> Rescher, N.:
Aspects of Action,
Rescher, N. (Hrsg.): The Logic of Decision and Action,
University of Pittsburgh Press, Pittsburgh, 1967, S.215-220.
- <Reu81> Reuter, A.:
Fehlerbehandlung in Datenbanksystemen,
Carl Hanser Verlag, München, 1981.
- <Rol81> Rolland, C.:
Some Concepts for an Information Systems Architecture,
Proc. of the ACM - European Regional Conference,
"Systems Architecture", Westbury House, London, 1981, S.482-492.
- <RS79> Rowe, L.A.; Shoens, K.A.:
Data Abstraction, Views and Updates in RIGEL,
Memorandum No. UCB/ERL M79/5, University of California,
Berkley, 1979.
- <Schl82> Schlörner, J.:
Outputkontrollen zur Sicherung statistischer Datenbanken
- Ein Überblick,
Informatik Spektrum, Band 5 (1982) Heft 4, S.224-236.
- <Schm78> Schmidt, J.W.:
Type Concepts for Database Definition,
Shneiderman, B. (Hrsg.): Databases: Improving Usability and
Responsivness, Academic Press, New York, 1978, S.215-244.

- <Schm80> Schmidt, J.W.:
PASCAL/R Report, Bericht Nr. 66,
Fachbereich Informatik, Universität Hamburg, 1980.
- <SB83> Schmidt, J.W., Brodie, M.L.:
Relational Database Systems, Analysis and Comparison,
Springer Verlag, Heidelberg, 1983.
- <Schu81> Schuster, S.A.:
Relational Data Base Management For On-Line Transaction
Processing, TANDEM Technical Report TR 81.5,
TANDEM Computers Inc., Cupertino, 1981.
- <Schw76> Schwemmer, O.:
Theorie der rationalen Erklärung - Zu den methodischen
Grundlagen der Kulturwissenschaften,
Verlag C.H.Beck, München, 1976.
- <Schw79> Schwemmer, O.:
Praxis, Methode und Vernunft: Probleme der Moralbegründung,
Zeitschrift für Pädagogik, 15.Beiheft, 1979, S.87-102.
- <SWL77> Shaw, M.; Wulf, W.; London, R.L.:
Abstraction and Verification in Alphas, Defining and
Specifying Iteration and Generators,
CACM, Vol.20 (1977), No.8, S.553-563.
- <SS77> Smith, J.M.; Smith, D.C.P.:
Database Abstractions: Aggregation and Generalization,
ACM TODS, Vol.2 (1977), No.2, S.105-133.
- <Sto75> Stonebraker, M.:
Implementation of Integrity Constraints and Views by Query
Modification, W.F. King (Hrsg.): Proc. Intern. Conf. on
Management of Data, San Francisco, 1975, S.65-78.
- <Sto76> Stonebraker, M.:
The Design and Implementation of INGRES,
ACM TODS, Vol.1 (1976), No.3.
- <SJR81> Stonebraker, M.; Johnson, R.; Rosenberg, S.:
Extending INGRES With a Rules System,
Memorandum No. UCB/ERL M81/93, University of California,
Berkley, 1981.
- <Thu80> Thurnherr, B.:
Konzepte und Sprachen für den Entwurf konsistenter Datenbanken,
Dissertation an der ETH-Zürich, Diss. ETH Nr. 6526, 1980.

- <TL76> Tsichritzis, D.; Lochovsky, F.:
Views on Data,
Jardine D.A. (Hrsg.): Proc. 2nd SHARE Conf. on DBMS,
Montreal, 1976, S.51-66.
- <U1180> Ullman, J.D.:
Principles of Database Systems,
Pitman Publishing, London, 1980.
- <VRT82> Vinek, G.; Rennert, P.F.; Tjoa, A M.:
Datenmodellierung - Theorie und Praxis des Datenbankentwurfs,
Physica Verlag, Würzburg - Wien, 1982.
- <Web78> Weber, H.:
A Software Engineering View of Data Base Systems,
Proc. 4th Conf. on VLDB, West Berlin, 1978, S.36-51.
- <Web81> Weber, W.:
Ein Subsystem zur Aufrechterhaltung der semantischen
Integrität in Datenbanken,
Jochen Heinzmann Verlag, Karlsruhe, 1981.
- <Wed79> Wedekind, H.:
System-Independent Program Design with Databases,
P.A. Samet (Hrsg.): Proc. EURO-IFIP 1979, London,
North-Holland Publ. Comp., S.65-70.
- <Wed79a> Wedekind, H.:
Eine Methodologie zur Konstruktion des Konzeptionellen Schemas,
Niedereichholz, J. (Hrsg.): Datenbanktechnologie,
B.G. Teubner, Stuttgart, 1979, S.65-79.
- <Wed80> Wedekind, H.:
Erweiterung einer Entwicklungsmethodologie für Datenbanksysteme
um eine Komponente zum Schutze vor Mißbrauch von personen-
bezogenen Daten, Angewandte Informatik, (1980) 7, S.255-266.
- <W080> Wedekind, H.; Ortner, E.:
Systematisches Konstruieren von Datenbankanwendungen -
Zur Methodologie der Angewandten Informatik,
Berichte zur praktischen Informatik, 16,
Carl Hanser Verlag, München, 1980.
- <Wed81> Wedekind, H.:
Datenbanksysteme I (2.Aufl.),
Reihe Informatik/16, B.I.-Wissenschaftsverlag, Mannheim, 1981.
- <Wed82> Wedekind, H.:
Nachlese zum Thema "Hierarchisierung und Modularisierung",
Softwaretechnik-Trends, Heft 2-3, 1982, S.45-51.

- <Wed83> Wedekind, H.:
Über die Grundlagen einer interaktiven Dialogführung in der
Datenerfassung, Angewandte Informatik (1983) 2, S.68-77.
- <Wed83a> Wedekind, H.:
Datenintegrität der Anwendung aus wissenschaftstheoretischer
Sicht, (noch nicht veröffentlicht).
- <WW79> Weinberger, Ch.; Weinberger, O.:
Logik, Semantik, Hermeneutik, Verlag C.H.Beck, München, 1979.
- <Wel82> Weller, D.L.:
A Relational Representation of an Abstract Type System,
IBM Research Report, RJ3503, San Jose, 1982.
- <Wir82> Wirth, N.:
Programming in Modula-2, Springer Verlag, Heidelberg, 1982.
- <Wri77> Wright, G.H.v.:
Handlung, Norm und Intention - Untersuchungen zur deontischen
Logik, de Gruyter Verlag, Berlin, 1977.
- <Wri79> Wright, G.H.v.:
Das menschliche Handeln im Lichte seiner Ursachen und Gründe,
Lenk, H. (Hrsg.): Handlungstheorien - interdisziplinär,
Band 2, Wilhelm Fink Verlag, München, 1979, S.417-430.
- <Yor76> Yormark, B.:
The ANSI/X3/SPARC/SGDMS Architecture,
Jardine D.A. (Hrsg.), Proc. 2nd SHARE Conf. on DEMS,
Montreal, 1976, S.1-22.
- <Zör82> Zörntlein, G.:
Untersuchungen zur Realisierung von Änderungsoperationen für
Externe Schemata von Datenbanken,
Studienarbeit am IMMD(VI), Universität Erlangen, 1982.
- <Zör83> Zörntlein, G.:
Fertigungsdatenbank auf ADABAS/M - Entwurf und Implementierung
einer Komponente für den Materialfluß,
Diplomarbeit am IMMD(VI), Universität Erlangen, 1983.

Anhang 1

Beispiel: Datenbank - Pharmaaußendienst

Das Konzeptionelle Schema dieser Beispieldatenbank umfaßt nur einen Teil der für einen Pharmaaußendienst notwendigen Grundbegriffe. Der Ausschnitt wurde aber so gewählt, daß er genügend komplex ist, um die wesentlichen Aspekte der in der Arbeit vorgetragenen Konzepte zu illustrieren.

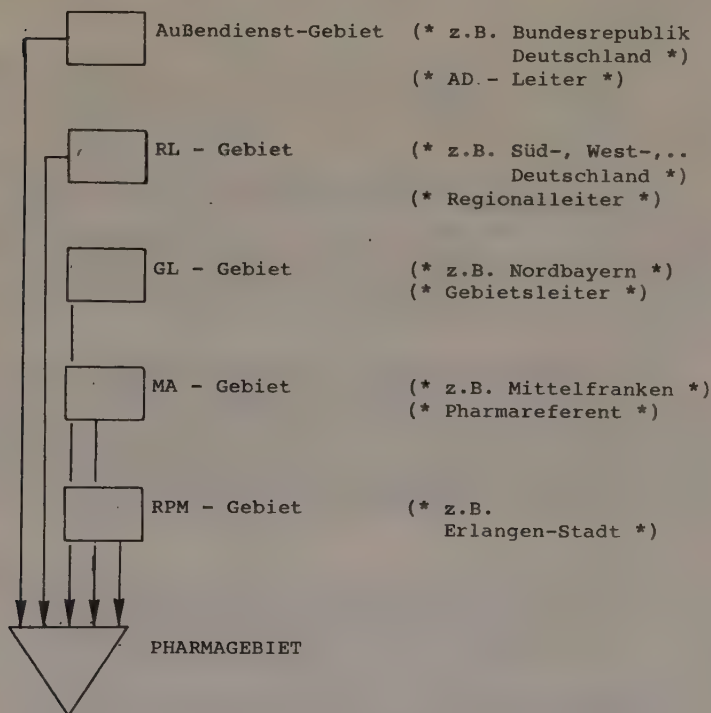
Der hier spezifizierte Ausschnitt der Datenbank umfaßt den wissenschaftlichen Außendienst eines pharmazeutischen Unternehmens. Es wird vorweggenommen, daß die im Land verteilten Außendienstmitarbeiter auf eine Datenbank ihrer Firma zugreifen können und über diese den Geschäftsverkehr mit der Zentrale und untereinander abwickeln. Der hier vorgestellte Anwendungsbereich wird bis dato meist dezentral und manuell durch den einzelnen (Außendienst-)Mitarbeiter abgewickelt, die Kommunikation erfolgt durch ein ausgereiftes Formularwesen über den Postweg. Ergänzende Teilaufgaben werden im Batch-Betrieb durch Erfassung der Daten in maschinenlesbaren Formblättern realisiert.

Die im Anhang verwendete Notation verbindet Elemente von QUEL <RT82> und Modula-2 <Wir82>, <RZ82>. Bei der Einbettung von QUEL in Modula-2 wurde auf die Verwendung von Steuerzeichen zur Markierung der QUEL-Ausdrücke verzichtet. Hinzu kommen Erweiterungen der in diesen Sprachen zur Verfügung stehenden Sprachkonstrukte, die sich aus der Arbeit ergeben.

(i) Die Gliederung des Gebiets des Außendienst in organisatorische Einheiten.

Abb. A1-1:

(* Konstruktionsdiagramm für die organisatorischen Gebietseinheiten *)



Die einzelnen organisatorischen Einheiten lassen sich durch Subordination zusammenfassen, so daß es genügt in der konkreten Datenbank nur die Relation PHARMAGEBIET zu realisieren.

(Die Relationenschemata wurden vereinfacht, daß sie zwar die für den Außendienst relevanten Teile umfassen, andererseits aber nicht durch ihren Umfang die Ausdruckskraft der gegebenen Beispiele schwächen.)

(Die Notation erfolgt in Anlehnung an Kapitel 3, <RZ82>, <Wir82>.)

```

-----
1                                     1
1  TYPE PHARMAGEBIET = RELATION OF RECORD 1
1                                     1
1                                     (* Beispiel *) 1
1           PhGeNr : CARDINAL;      (* 27 *) 1
1  Bezeichnung : NAME;              (* Mittelfranken *) 1
1  Gebietstyp : (AD,RL,GL,MA,RPM);  1
1                                     (* MA *) 1
1           OrgEinheit : PHARMAGEBIET.PhGeNr; 1
1                                     (* 07 *) 1
1                                     1
1  .... 1
1                                     1
1  IDENT = (PhGeNr) 1
1  END; 1
1                                     1
-----

```

Die Punkte ("....") in den Relationenschemata stehen für weitere Attribute, die für ein Informationssystem eines pharmazeutischen Unternehmens relevant sind, aber nichts zur Illustration im betrachteten Zusammenhang beitragen würden. (In "(* .. *)" werden zur Erläuterung typische Werte angegeben.)

Die Angabe von "Grundtypen" bezieht sich auf die in Modula-2 (<Wir82>) verfügbaren Datentypen. Im Relationenschema können aber auch Aufzählungstypen und "flache" Recordtypen angegeben werden. (Flach bedeutet keine tiefere Schachtelung, so daß nach dem Relationalen Datenmodell, die Komponenten als eigentliche Attribute aufzufassen sind.) Zur Angabe der Datentypen vergl. Abs. 5.3.1.

In der Relation PHARMAGEBIET kommt der Datentyp NAME vor.

TYPE NAME = ARRAY [0..30] OF CHAR;

Neben den strukturellen Integritätsbedingungen ist die Angabe von semantischen Bedingungen im Konzeptionellen Schema erforderlich. Exemplarisch werden im folgenden zu den Relationen einige Bedingungen aufgeführt. Zur Notation der semantischen Integritätsbedingungen vergleiche Abs. 5.3.2.

Als Bedingung für die Pharmagebiete ergibt sich (die nicht durch die referentielle Bedingung abgedeckt wird):
"Die Gebietseinheiten sind hierarchisch gegliedert."

```
RANGE OF gebiet1 IS PHARMAGEBIET;
RANGE OF gebiet2 IS PHARMAGEBIET;
DEFINE
ASSERTION ON PHARMAGEBIET IS
    ((gebiet2.Gebietstyp)
      WHERE gebiet2.PhGeNr = gebiet1.OrgEinheit
      AND gebiet1.Gebietstyp = "RPM")
    = "MA";
:
:
```

```
RANGE OF gebiet IS PHARMAGEBIET
DEFINE
ASSERTION ON PHARMAGEBIET IS
    (gebiet.OrgEinheit
      WHERE gebiet.Gebietstyp = "AD")
    = NIL;
```

(ii) Die Basisrelationen ARZT, MITARBEITER und MEDIKAMENT

```

-----
|
| TYPE ARZT = RELATION OF RECORD          (* Beispiel *)
|
|   ArztNr : CARDINAL;                    (* 4711 *)
|   Anrede : (Frau, Herr);                (* Herr *)
|   Titel  : TITEL;                       (* Dr. med *)
|   Vorname: NAME;                        (* Axel *)
|   Name   : NAME;                        (* Krause *)
|   Adresse: ADRESSE;                     (* Schillerstr. 13
|                                     8800 Ansbach *)
|   Telefon: TELEFON;                     (* 0981/343536 *)
|   Fach   : SPARTE;                      (* Internist *)
|   Zusatz : ZUSATZSPARTE;                (* Kardiologe *)
|   NebenTaetig : NEBENTAETIG;            (* Betriebsarzt *)
|   Kategorie : KATEGORIE;                (* Kasse *)
|   Klasse  : KLASSE;                     (* A *)
|   Geb-Datum : DATUM;                    (* 29. 3.40 *)
|   NL-Jahr  : [00..99] OF CARDINAL;      (* 76 *)
|   Besuchszeit : BESUCHSZEITEN;          (* n.v. *)
|   B-Wunsch  : NOTIZ;                    (* Termin *)
|   Aussendung : (ja, nein);              (* ja *)
|   Bemerkungen : NOTIZ;                  (* Preisbewußt *)
|   Interessen : NOTIZ;                   (* Afrikareisen *)
|   RPM-KreisNr : PHARMAGEBIET.PhGeNr;
|               (* importiert aus Relation PHARMAGEBIET
|               vergleiche Abs. 5.3.1 *)
|   ...
|
|   IDENT = (ArztNr)
|   END;
|
-----

```

In der Relation ARZT kommen folgende nichtelementare Datentypen vor.

```
TYPE TITEL = ( Dr. med, Dr., Prof. Dr., PD Dr. med, ... );
```

(In Modula-2 werden die Subtypen (vgl. NL-Jahr in ARZT) einfach durch ihre Grenzen spezifiziert, diese Vorgehensweise setzt voraus, daß ein Zeichen für alle Datentypen eindeutig ist. Für Datenbanken ist es jedoch wünschenswert, daß ein Element in verschiedenen Typen angegeben werden kann. Es wird deshalb hier bei der Vereinbarung von Subtypen die zusätzliche Angabe des Basistyps gemacht.)

Beispiel: Datenbank "Pharmaaußendienst"

TYPE ADRESSE = RECORD

Strasse : NAME;

HausNr : CARDINAL;

PLZ : [1000 .. 9999] OF CARDINAL;

Wohnort : NAME

END;

TYPE TELEFON = RECORD

Vorwahl : CARDINAL;

const : (/);

Nr : CARDINAL

END;

TYPE SPARTE = (Internist, Dermatologe, Urologe,
Praktischer Arzt, ...);TYPE ZUSATZSPARTE = (Kardiologe, Allergologe,
Gastroentrologe, ...);TYPE NEBENTAETIG = (Betriebsarzt, Prüfarzt-Konkurrenz,
Belegarzt, ...);

TYPE KATEGORIE = (Kasse, Privat, Klinik, ...);

(* "... " in den Aufzählungstypen bedeuten, daß nur ein
Ausschnitt angeben *)

TYPE KLASSE = (A, B, C, N);

(* Klasse über die Praxisgröße $A > B > C > N$,
N steht für Neueröffnungen *)

TYPE DATUM = RECORD

Tag : [0 .. 31] OF CARDINAL;

c1 : (.);

Monat : [0 .. 12] OF CARDINAL;

c2 : (.);

Jahr : [0 .. 99] OF CARDINAL

END;

(* Auf dem Grundtyp DATUM gibt es Addition und Sub-
traktion von Tagen, Monaten und Jahren. Über
den Typ gilt die kalendarische Ordnungsrelation *)

TYPE BESUCHSZEITEN = ...

(* Auf eine Angabe dieses umfangreichen, aber sonst
nebensächlichen Recordtyps wird verzichtet *)

TYPE NOTIZ = ARRAY [0..100] OF CHAR;

Als eine spezifische Ingetritätsbedingung für den Arzt ist die Zugehörigkeit des Orts seiner Praxis zu einem RPM-Kreis. Diese Bedingung würde aber eine Verwaltung aller Orte erfordern, die hier nicht sinnvoll erscheint.

Eine weitere Bedingung ist die Einschränkung des Wertebereichs für die Pharmagebiete in der Relation ARZT auf RPM-Gebiete.

```
RANGE OF arzt IS ARZT;
RANGE OF gebiet IS PHARMAGEBIET;
DEFINE
ASSERTION ON ARZT, PHARMAGEBIET IS
  ((gebiet.Gebietstyp)
    WHERE arzt.RPM-KreisNr = gebiet.PhGeNr)
  = "RPM"
```

Als finale deontische Integritätsbedingungen (vgl. Abs. 5.3.3) sind für die Verwaltung der Ärztestammdaten einige subjektorientierte Anforderungen angebbbar. Sie werden im folgenden wieder aufgegriffen und es wird gezeigt, wie sie bereits auf den einzelnen höheren Ebenen eines Datenbank-Anwendungssystems behandelt werden können.

Es darf z.B. der Pharmareferent die Ärzte seines Gebiets bearbeiten.

(Der Zusatz "-OLD" bzw. "-NEW" gibt an, ob auf die Relation vor oder nach Ausführung der Zustandsänderung Bezug genommen wird. Relationen ohne Zusatz bleiben unverändert.)

```
RANGE subject IS MITARBEITER;
RANGE arztold IS ARZT-OLD;
RANGE arztnew IS ARZT-NEW;
RANGE gebiet IS PHARMAGEBIET;

DEFINE PERMISSION ON ARZT
  FOR ((subject.MA-Nr)
    WHERE subject.Funktion = "Pharmareferent"
  TRANSITION
  FROM
    TRUE
  TO
    ((arztold.all)
      WHERE arztold.RPM-KreisNr = gebiet.PhGeNr
      AND gebiet.OrgEinheit = subject.MA-KreisNr)
    !=
    ((arztnew.all)
      WHERE arztnew.RPM-KreisNr = gebiet.PhGeNr
      AND gebiet.OrgEinheit = subject.MA-KreisNr)
```

:

Allen Mitarbeitern, die nicht Pharmareferenten oder Gebietsleiter sind, ist die Veränderung von Ärztestammdaten untersagt.

```
DEFINE PROHIBITION ON ARZT
FOR ((subject.MA-Nr)
      WHERE subject.Funktion != "Pharmareferent"
      OR subject.Funktion != "Gebietsleiter"
TRANSITION
FROM TRUE TO TRUE;
(* Das bedeutet jeder Übergang ist für alle Zustände
   verboten *)
```

Die zweite elementare personenbezogene Relation:

```
-----
|
| TYPE MITARBEITER = RELATION OF RECORD
|
| MA-Nr : CARDINAL; (* Beispiel *)
| Anrede : (Frau, Herr); (* Frau *)
| Funktion : FUNKTION; (* Pharmareferent *)
| Vorname : NAME; (* Petra *)
| Name : NAME; (* Schulze *)
| Adresse : ADRESSE; (* Wagnerstr. 167
| 851 Fürth *)
| Telefon : TELEFON; (* 0911/323130 *)
| MA-KreisNr : PHARMAGEBIET.PhGeNr;
| (* importiert aus Relation PHARMAGEBIET
| vergleiche Abs. 5.3.1 *)
| PKW : KFZ.Kennzeichen;
| (* importiert aus Relation KFZ
| nicht weiter aufgeführt *)
| ....
|
| IDENT = (MA-Nr)
| END;
|
|-----
```

Eine semantische Integritätsbedingung ist, daß die "Funktion" des Mitarbeiters mit dem ihm zugeordneten Gebietsverträglich ist. Pharmareferenten ist z.B. ein Mitarbeitergebiet zugeordnet.

Für die Relation MITARBEITER ergeben sich ganz ähnliche finale deontische Bedingungen wie bei der Relation ARZT, allerdings für den hier nicht betrachteten Bereich des Personalwesens.

Die Relation MEDIKAMENT enthält alle Angaben über die
Medikamente des Unternehmens:

```

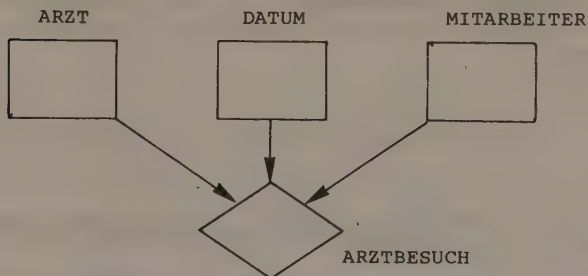
1 TYPE MEDIKAMENT = RELATION OF RECORD
1
1 (* Beispiel *)
1 Name : NAME; (* BetaBlock *)
1
1 Indikation : INDIKATIONSINDEX; (* C7A *)
1
1 Produktmanager : MITARBEITER.MA-Nr; (* 4913 *)
1
1 Einfuehrung : DATUM; (* 11.11.71 *)
1
1 Preis : BETRAG; (* 27.50 *)
1
1 ApothekenPreis : BETRAG; (* 54.20 *)
1
1
1 ....
1
1 IDENT = (Name)
1
1 END;
1

```

- (iv) Zusammengesetzte Begriffe um den Besuch eines Außendienst-Mitarbeiters bei einem Arzt zu beschreiben.

Abb. A1-2.:

(* Konstruktionsdiagramm für die Relation ARZTBESUCH *)



```

!-----!
! TYPE ARZTBESUCH = RELATION OF RECORD                                     !
!                                                                                   !
!           MA-Nr : MITARBEITER.MA-Nr;      (* Beispiel *)                   !
!           ArztNr : ARZT.ArztNr;           (* 4813 *)                       !
!           Datum  : DATUM;                 (* 4712 *)                       !
!           FolgeNr : CARDINAL;              (* 2. 5. 83 *)                   !
!           (* Sequenz der besuchten Ärzte *)                                     !
!           Bemerkung : NOTIZ;                (* keine Wartezeit *)           !
!           Begleitung : MITARBEITER.MA-Nr;   (* 4914 *)                       !
!                                                                                   !
!           ....                                                                    !
!           IDENT = (MA-Nr,ArztNr,Datum)                                           !
!           END;                                                                    !
!-----!

```

Eine semantische Integritätsbedingung ergibt sich über die Sequenz der Arztbesuche (Mitarbeiter und Tages spezifisch)

Beispiel: Datenbank "Pharmaaußendienst"

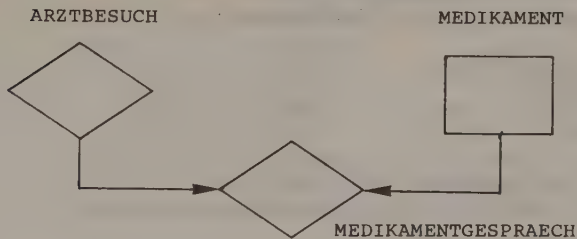
```

RANGE OF besuch1 IS ARZTBESUCH;
RANGE OF besuch2 IS ARZTBESUCH;
DEFINE
ASSERTION ON ARZTBESUCH IS
    COUNT (besuch1.FolgeNr
            BY besuch1.MA-Nr, besuch1.Datum)
    =
    (COUNTU (besuch1.FolgeNr
              BY besuch1.MA-Nr, besuch1.Datum
              WHERE besuch1.FolgeNr = besuch2.FolgeNr +1
              AND besuch2.MA-Nr = besuch1.MA-Nr
              AND besuch2.Datum = besuch1.Datum)
    + 1)

```

Abb. A1-3.:

(* Konstruktionsdiagramm für die Relation
 MEDIKAMENTGESPRACH *)



```

-----
| TYPE MEDIKAMENTGESPRACH = RELATION OF RECORD |
|   MA-Nr : MITARBEITER.MA-Nr; |
|   ArztNr : ARZT.ArztNr; |
|   Datum : DATUM; |
|   Medikament : MEDIKAMENT.Name; |
|   Bemerkung : NOTIZ; |
|   Muster : [0..4] OF CARDINAL; |
|   Besprochen : (ja,nein); |
|   .... |
| IDENT = (MA-Nr,ArztNr,Datum,Medikament) : |
| END; |
|-----

```

Beispiel: Datenbank "Pharmaaußendienst"

Als Integrität ist die Existenz eines zugehörigen Arztbesuchs zu fordern.

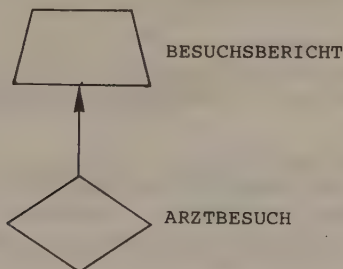
```
RANGE OF besuch IS ARZTBESUCH;
RANGE OF gespraech IS MEDIKAMENTGESPRACH;
DEFINE
ASSERTION ON ARZTBESUCH, MEDIKAMENTGESPRACH IS
  ( besuch.MA-Nr, besuch.Datum, besuch.ArztNr
    BY besuch.MA-Nr, besuch.Datum)
>
  ( gespraech.MA-Nr, gespraech.Datum, gespraech.ArztNr
    BY gespraech.MA-Nr, gespraech.Datum)
```

Als eine finale deontische Integritätsbedingung ist anzugeben, daß für einen Pharmareferenten der Besuch von mehr als 8 Ärzten explizit erlaubt ist. Bei einem geringern Besuchsangabe könnte man in Abhängigkeit von Gründen der Anrechnung in Besuchsbericht einen Trigger formulieren.

```
RANGE OF subject IS MITARBEITER;
RANGE OF besuch IS ARZTBESUCH;
RANGE OF gespraech IS MEDIKAMENTGESPRACH;
DEFINE
PERMISSION ON ARZTBESUCH
  FOR (subject.MA-Nr
    WHERE subject.Funktion = "Pharmareferent")
  TRANSITION
  FROM
    ( COUNT (besuch.MA-Nr BY besuch.Datum
      WHERE besuch.MA-Nr = subject.MA-Nr)
    = 0 )
  AND
    besuch.Datum = (DATE - 2 DAYS)
  TO
    COUNT (besuch.MA-Nr, BY besuch.Datum
      WHERE gespraech.ArztNr = besuch.MA-Nr
      AND gespraech.MA-Nr, = besuch.MA-Nr
      AND gespraech.Datum = besuch.Datum)
  >= 8
```

Abb. A1-4.:

(* Konstruktionsdiagramm der Relation BESUCHSBERICHT *)



```

-----
!
! TYPE BESUCHSBERICHT = RELATION OF RECORD
!
!           MA-Nr : MITARBEITER.MA-Nr; (* Beispiel *)
!           Datum : DATUM;             (* 2. 5.83 *)
!           km-dienst : CARDINAL;       (* 243 *)
!           km-privat : CARDINAL;       (* 12 *)
!           Abfahrt : UHRZEIT;          (* 7.00 *)
!           Ankunft : UHRZEIT;          (* 17.35 *)
!           Bemerkung : NOTIZ;           (* - *)
!           Anrechnung : 0.0, 0.5, 1.0 OF REAL;
!                                   (* 0,5 *)
!           Begruendung : AUSFALL;      (* G-Tagung *)
!
!           ....
!           IDENT = (MA-Nr, Datum)
! END;
!
-----
  
```

Weitere Datentypen sind :

```

TYPE UHRZEIT = RECORD
    std : 0..24 ;
    c   : (.);
    min : 00..59
END;
  
```

```

TYPE AUSFALL = (G-Tagung, Gemeinschaftsbesuch,
               F-Tagung, R-Tagung, AD-Tagung,
               Unfall, Krankheit, ... )
  
```

Wenn der Anrechnungsfaktor kleiner 1.0 ist, ist hierfür eine Begründung anzugeben.

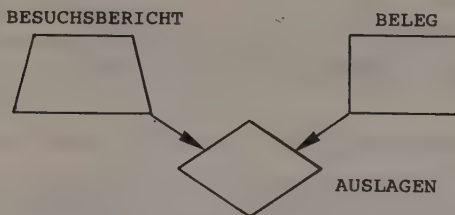
```
RANGE OF bericht IS BESUCHSBERICHT;  
DEFINE  
ASSERTION ON BESUCHSBERICHT IS  
  (( bericht.ALL) WHERE bericht.Anrechnung < 1.0  
    AND bericht.Begrundung = NIL)  
= EMPTY
```

Weitere Integritätsbedingungen wären z.B. Angaben über die Uhrzeit, ob es möglich ist in der angegebenen Zeit entsprechend viele Ärzte zu besuchen. Auch die gefahrenen Kilometer könnten anhand der besuchten Ärzte kontrolliert werden, wenn man eine entsprechende Verwaltung der Orte noch vornehmen würde.

- (v) Die Relationen sind damit noch nicht erschöpft.
Es fehlen z.B. noch Informationstechnischer Art, wie die Auslagen eines Mitarbeiters.

Abb. A1-5.:

(* Konstruktionsdiagramm der Relation AUSLAGEN *)



```

-----
!
! TYPE AUSLAGEN = RELATION OF RECORD
!
!   MA-Nr : MITARBEITER.MA-Nr;
!   BelegNr : CARDINAL;
!   Datum : DATUM;
!   Bezeichnung : NAME;
!   Kostenart : KOSTENART;
!   Betrag : BETRAG;
!   ....
!
!   IDENT = (MA-Nr, Beleg, Datum)
! END;
!
-----

```

```

TYPE KOSTENART = ( Wagenpflege, Tanken, Parken, Werbung,
                  Tagesspesen, Bürobedarf, ... )

```

Als semantische Integritätsbedingungen ergeben sich z.B., daß die Kostenart "Tagesspesen" von der Abwesenheit des Mitarbeiters von seiner Wohnung abhängt.

```

RANGE OF auslagen IS AUSLAGEN;

```

```

RANGE OF bericht IS BESUCHSBERICHT;

```

```

DEFINE

```

```

ASSERTION ON AUSLAGEN, BESUCHSBERICHT IS

```

```

  (( auslagen.Betrag BY auslagen.MA-Nr, auslagen.Datum
    WHERE auslagen.Kostenart = Tagesspesen
    AND (bericht.Ankunft - bericht.Abfahrt)
    >= 10 HOURS ))

```

```

= 36.00 DM

```


Beispiel: Datenbank "Pharmaaußendienst"

(vi) Weitere Relationen für den Pharma-Außendienst sind
z.B.:

VERANSTALTUNG (Bezeichnung, Typ, Von, Bis,
Ort, Verantwortlich, Bemerkung, ...)

VERANSTALTUNGSBETREUUNG (Veranstaltung, Mitarbeiter,
Von, Bis, ...)

BESUCHSGROBPLANUNG (MA-Nr, Datum, Pharmakreis, ...)

ARZTTERMIN (MA-Nr, Datum, Zeitpunkt, ArztNr)

WAGENZUSTANDSBERICHT (KFZ, km-Stand, Kundendienst, ...)

Im folgenden werden statische und operationale Sichten auf die in Anhang 1 angegebene Pharmadatenbank, wie sie für die Tätigkeit eines Pharmareferenten nötig sind, gebildet. Das Hauptkriterium ist hierbei die Beschränkung des Ausschnitts der Datenbank auf die niedergelassenen Ärzte seines Gebiets. Es werden im statischen Teil, der Beschreibung der Sichten die abgeleiteten Relationen nur insoweit angegeben, wie sie nicht nach dem gleichen Prinzip gebildet sind. Analoges gilt für die Operationen auf den Sichten.

Die Angabe erfolgt entsprechend dem in Abs. 3.1 vorge-
stellten Rahmen:

- Abb. 2-1.:

(* Externes Schema ARZT *)

!		!
!	ARZT	!
!		!



```

!                                     !
!   ARZT   ARZTBESUCH               !
!                                     !

```

```
EXTERNAL-SCHEMA AERZTEBESTAND (Mitarbeiter);
(* Definition einer virtuellen Datenbank zur Verwaltung
  des Ärztebestands eines Außendienstmitarbeiter *)
```

Beispiel: Sicht auf die "Pharmadatenbank"

SPECIFICATION;

(* Angabe der exportierten abgeleiteten Relationen *)

EXPORT PERSISTENT ARZT;

(* Angabe der zum Aufbau der Sicht benötigten Relationen
aus der Datenbank *)

FROM DATABASE PHARMADATENBANK IMPORT PERSISTENT
ARZT, ARZTBESUCH, MITARBEITER, PHARMAGEBIET;

(* Festlegen der Struktur der Abgeleiteten Relationen *)

TYPE ARZT = VIEW OF RECORD

ArztNr : CARDINAL;

:

Name : NAME;

:

Fachgebiet : SPARTE;

:

RPM-KreisNr : CARDINAL;

letzterBesuch : DATUM;

Besuchsbemerkung : NOTIZ;

AnzahlBesuche : CARDINAL;

IDENT = (ArztNr)

END;

(* Die abgeleitete Relation ARZT wurde um die einige
Attribute aus ARZTBESUCH erweitert, um das Beispiel
interessanter zu gestalten *)

REPRESENTATION;

(* Festlegen der Wertebereiche der Tupelvariablen *)

(* für die abgeleiteten Relationen *)

RANGE OF arztview IS ARZT VIEW;

(* für die Relationen aus der Datenbank *)

RANGE OF arzt IS ARZT RELATION;

RANGE OF gebiet IS PHARMAGEBIET RELATION;

RANGE OF mitarbeit IS MITARBEITER RELATION;

RANGE OF besuch1 IS ARZTBESUCH RELATION;

RANGE OF besuch2 IS ARZTBESUCH RELATION;

(* Restriktion der Ärzte auf das Mitarbeitergebiet *)

DEFINE VIEW ARZT

(ArztNr=arzt.ArztNr, ..., RPM-Kreisnr=arzt.RPM-KreisNr,
letzterBesuch=

MAX (besuch2.datum BY arzt.ArztNr

WHERE arzt.ArztNr = besuch1.ArztNr

AND besuch2.MA-Nr = Mitarbeiter)

Besuchsbemerkung=besuch1.Bemerkung,

Beispiel: Sicht auf die "Pharmadatenbank"

```

AnzahlBesuche=
    COUNT (besuch2.ArztNr BY arzt.ArztNr
           WHERE besuch2.ArztNr = arzt.ArztNr
             AND besuch2.Datum.Jahr >= DATE.Jahr
             AND besuch2.MA-Nr = Mitarbeiter))
WHERE besuch1.ArztNr = arzt.ArztNr
  AND mitarbeit.MA-Nr = Mitarbeiter
  AND gebiet.OrgEinheit = mitarbeit.MA-KreisNr
  AND gebiet.PhGeNr = arzt.RPM-KreisNr);

OPERATION;
(* Festlegen der Übersetzung der Sicht-Operationen *)
(* vgl. Abs. 3.1 *)
DEFINE ON ARZT VIEW APPEND, DELETE
  BY STANDARD ON ARZT RELATION;
(* Die aus ARZTBESUCH abgeleiteten Attribute werden
   ignoriert *)

DEFINE ON ARZT VIEW
  REPLACE (Anrede=arztview.Anrede, ...,
           Interessen=arztview.Interessen,
           RPM-KreisNr=arztview.RPM-KreisNr,
           Besuchsbemerkung=arztview.Besuchsbemerkung)
BY ON ARZT RELATION
  REPLACE (Anrede=arztview.Anrede, ...,
           Interessen=arztview.Interessen,
           RPM-KreisNr=arztview.RPM-KreisNr)
  WHERE arzt.ArztNr = arztview.ArztNr
ON ARZTBESUCH RELATION
  REPLACE (Bemerkung=arztview.Besuchsbemerkung)
  WHERE besuch2.ArztNr=arztview.ArztNr
    AND besuch2.datum = MAX (besuch1.dat
    BY arzt.ArztNr
    WHERE arztview.ArztNr = besuch1.ArztNr);

SCHEMA-END AERZTEBESTAND.

```

(Verzichtet man auf die Information aus der Relation ARZTBESUCH in der abgeleiteten Sicht, so vereinfacht sich die Sichtbildungsfunktion wesentlich. An diesem Beispiel sollte die Verwendung eingebetteter Funktionen bei der Sichtdefinition veranschaulicht werden.

Beispiel: Sicht auf die "Pharmadatenbank"

- (ii) Beispiel: ARZTBESUCH und MEDIKAMENTGESPRACH
(Die abgeleiteten Relationen werden um Attribute aus der Relation ARZT erweitert.)

Abb. A2-2.:

(* Externes Schema ARZTBESUCH und MEDIKAMENTGESPRACH *)

Sicht:

	ARZTBESUCH	MEDIKAMENTGESPRACH



Datenbank:

	ARZTBESUCH	ARZT	MEDIKAMENTGESPRACH

EXTERNAL-SCHEMA ARZTBESUCHSBERICHTE (Mitarbeiter);
(* Definition einer virtuellen Datenbank für die Berichte eines Pharmaaußendienstmitarbeiters *)

SPECIFICATION;

(* Angabe der exportierten abgeleiteten Sichten *)

EXPORT PERSISTENT

ARZTBESUCH, MEDIKAMENTGESPRACH;

(* Angabe der zum Aufbau der Sicht benötigten Relationen aus der Datenbank *)

FROM DATABASE PHARMADATENBANK IMPORT PERSISTENT

ARZT, ARZTBESUCH, MEDIKAMENTGESPRACH

MITARBEITER;

(* Festlegen des Aufbaus der Abgeleiteten Relationen *)

TYPE ARZTBESUCH = VIEW OF RECORD

ArztNr : ARZT.ArztNr;

ArztName : ARZT.Name;

Datum : DATUM;

Bemerkung : NOTIZ;

Begleitung : CARDINAL;

Begl-Name : NAME;

IDENT = (ArztNr, Datum)

END;

(* Der komplexe Schlüssel aus MA-Nr, ArztNr, Datum reduziert sich durch die Abbildungsfunktion *)

Beispiel: Sicht auf die "Pharmadatenbank"

```
TYPE MEDIKAMENTGESPRAECH = VIEW OF RECORD
    ArztNr : ARZT.ArztNr;
    ArztName : ARZT.Name;
    Datum : ARZT.Datum;
    Medikament : NAME;
    Musterabgabe : 0 .. 4 OF CARDINAL;
    Besprochen : (ja,nein);
    Bemerkung : NOTIZ;
    IDENT = (ArztNr,Datum,Medikament)
END;

REPRESENTATION;
(* Restriktion der Aerzte auf Mitarbeitergebiet *)

(* Festlegen der Wertebereiche für die Tupelvariablen *)
(* Für die Relationen aus der Datenbank *)
RANGE OF arzt      IS ARZT      RELATION;
RANGE OF mitarbeit IS MITARBEITER RELATION;
RANGE OF besuch    IS ARZTBESUCH RELATION;
RANGE OF gespraech IS MEDIKAMENTGESPRAECH RELATION;

DEFINE VIEW ARZTBESUCH
    (ArztNr=besuch.ArztNr, ArztName=arzt.Name,
     Datum=besuch.Datum, Bemerkung=besuch.Bemerkung,
     Begleitung=besuch.Begleitung,
     Begl-Name=mitarbeit.Name)
    WHERE  besuch.MA-Nr = Mitarbeiter
    AND    besuch.ArztNr = arzt.ArztNr
    AND    besuch.Begleitung = mitarbeit.MA-Nr;

DEFINE VIEW MEDIKAMENTGESPRAECH
    (ArztNr=gespraech.ArztNr, ArztName=arzt.Name,
     Datum=gespraech.Datum, .... )
    WHERE  gespraech.MA-NR = Mitarbeiter
    AND    gespraech.ArztNr = arzt.ArztNr;

OPERATION;

DEFINE ON ARZTBESUCH VIEW
    BY STANDARD ON ARZTBESUCH RELATION;
:
DEFINE ON MEDIKAMENTGESPRAECH VIEW
    BY STANDARD ON MEDIKAMENTGESPRAECH RELATION;
(* STANDARD bedeutet entsprechend den in Abs. 3.1
   beschriebenen Mechanismen *)

SCHEMA-END ARZTBESUCHSBERICHTE.
```

(iii) Beispiel: Das Externe Schema der Dienstberichte
(Es wird durch Restriktion aus den Relationen
BESUCHSBERICHT und AUSLAGEN aus dem Konzeptionellen
Schema abgeleitet.)

Abb. A2-3.:

(* Externes Schema BESUCHSBERICHT und AUSLAGEN *)

Sicht:

	BESUCHSBERICHT AUSLAGEN



Datenbank:

!		!	
!	BESUCHSBERICHT	AUSLAGEN	!
!			!

EXTERNAL-SCHEMA DIENSTBERICHTE (Mitarbeiter);
(* Definition einer virtuellen Datenbank für die Berichte
eines Pharmaaußendienstmitarbeiters *)

SPECIFICATION;

(* Angabe der exportierten abgeleiteten Sichten *)

EXPORT PERSISTENT

BESUCHSBERICHT, AUSLAGEN;

(* Angabe der zum Aufbau der Sicht benötigten Relationen
aus der Datenbank *)

FROM DATABASE PHARMADATENBANK IMPORT PERSISTENT
BESUCHSBERICHT, AUSLAGEN;

(* Festlegen des Aufbaus der Abgeleiteten Relationen *)

TYPE BESUCHSBERICHT : VIEW OF RECORD

Datum : DATUM;

:

IDENT = (Datum)

END;

TYPE AUSLAGEN : VIEW OF RECORD

Beleg : CARDINAL;

Datum : DATUM;

:

IDENT (Beleg,Datum)

END;

Beispiel: Sicht auf die "Pharmadatenbank"

REPRESENTATION;

(* Festlegen der Wertebereiche für die Tupelvariablen *)

(* Für die Relationen aus der Datenbank *)

RANGE OF besuch IS BESUCHSBERICHT RELATION;

RANGE OF auslagen IS AUSLAGEN RELATION;

DEFINE VIEW BESUCHSBERICHT (ALL)

WHERE besuch.MA-Nr = Mitarbeiter;

DEFINE VIEW AUSLAGEN (ALL)

WHERE auslagen.MA-Nr = Mitarbeiter;

OPERATION;

DEFINE ON BESUCHSBERICHT VIEW

BY STANDARD ON BESUCHSBERICHT RELATION;

DEFINE ON AUSLAGEN VIEW

BY STANDARD ON AUSLAGEN RELATION;

(* STANDARD bedeutet entsprechend den in Abs. 2.3.1
beschriebenen Mechanismen *)

SCHEMA-END DIENSTBERICHTE.

Beispiel: Sicht auf die "Pharmadatenbank"

Teil 2:

Erweiterung zum Virtuelle Abstrakten Datentyp

Die einzelnen Externen Schemata sind zu Datenbank-Anwendungsmoduln zu erweitern. Dies wird exemplarisch an den Moduln ArztVerwaltung und ArztBesuchsVerwaltung vorgeführt.

(i) Spezifikation des Moduls "ArztVerwaltung"

(Er dient zur Bearbeitung der Ärzte-Stammdaten auf der Sicht für einen Außendienstmitarbeiter.

Es wurde dabei angenommen, daß die Wirtssprache mengenorientiert ist (vgl. Abs. 3.2).

Abb. A2-4.:

(* Datenbank-Anwendungsmodul "ArztVerwaltung" *)

DB-Modul:

```

-----
! AerzteEintragen,      !
! AerzteAustragen,     !
! AerzteAendern,       !
! AerzteSuchen         !
!                       !
! - - - - -           !
!                       !
!           ARZT       !
!                       !
!                       !
-----

```

DB-APPLICATION-MODULE

ArztVerwaltung (Mitarbeiter: CARDINAL);

SPECIFICATION;

```

EXPORT QUALIFIED AerzteEintragen, AerzteAustragen,
                  AerzteAendern, AerzteSuchen,
                  Aerzte, Arzt;

```

IMPORT PERSISTENT AERZTEBESTAND;

(* Sicht der Datenbank, wie in Teil 1 beschrieben *)

TYPE Arzt : RECORD OF

ArztNr : CARDINAL;

:

Name : NAME;

:

Beispiel: Sicht auf die "Pharmadatenbank"

```

        RPM-KreisNr : CARDINAL;
        letzterBesuch : DATUM;
        Besuchsbemerkung : NOTIZ;
        AnzahlBesuche : CARDINAL
END;
(* die anderen Attribute vergleiche Definition der Relation
   Arzt in Anhang 1 und Teil 1 zur abgeleiteten Relation *)

TYPE Aerzte = SET IN UNIVERSE OF Arzt;

VAR arztvar      : Arzt;
VAR arztmenge    : Aerzte;

RANGE OF arzt      IS ARZT-NEW;
RANGE OF arztold   IS ARZT-OLD;

ACTION AerzteEintragen
    (arztmenge: Aerzte, VAR aerztnr: NummernMenge);
(* analog zur Beschreibung in Abs. 3.2 *)
VAR NeueArztNr : SET IN Nummernmenge;
VAR neuearztnr : CARDINAL;
(* Die Arztnummer wird von der Aktion vergeben *)

PRE-CONDITION:
    (FORALL arztvar IN arztmenge: arztvar.ArztName != NIL
                                     AND arztvar.ArztNr   = NIL)
    (* es können noch weitere Komponenten in der Variablen
       "arztvar" unbesetzt sein.)
AND                                     (* CONDITION ON ARZT *)
    (FORALL neuearztnr IN NeueArztNr :
        NOT ANY (arztold.ArztNr)
                WHERE neuearztnr = arztold.ArztNr )
    (* Es gibt noch keinen Eintrag mit einen dieser Nummern *)

POST-CONDITION:
    (aerztenr = NeueArztNr)
AND
    (FORALL arztvar IN arztmenge :
        EXISTS1 neuearztnr IN NeueArztNr:
            arztvar.ArztNr = neuearztnr )
AND                                     (* CONDITION ON ARZT *)
    (FORALL neuearztnr IN NeueArztNr:
        COUNT (arzt.ArztNr WHERE neuearztnr = arzt.ArztNr
                                AND arztvar.Name = arzt.Name)
            = 1 )
AND                                     (* CONDITION ON ARZT *)
    (arzt.ALL) WHERE arztold.ArztNr = arzt.ArztNr
    =
    (arztold.ALL)
ACTION-END;
```

Beispiel: Sicht auf die "Pharmadatenbank"

ACTION AerzteAustragen (arztmenge: Aerzte);

PRE-CONDITION:

(FORALL arztvar IN arztmenge:

 arztvar.ArztNr != NIL AND arztvar.ArztName != NIL)

AND (* CONDITION ON ARZT *)

(FORALL arztvar IN arztmenge:

 ANY (arztold.ArztNr WHERE arztold.ArztNr = arztvar.ArztNr
 AND arztold.Name = arztvar.Name))

POST-CONDITION: (* CONDITION ON ARZT *)

(FORALL arztvar IN arztmenge :

 NOT ANY (arzt.ArztNr WHERE arzt.ArztNr = arztvar.ArztNr))

AND

((arztold.ALL) WHERE arztold.ArztNr != arzt.ArztNr)

=

(arzt.ALL)

ACTION-END;

ACTION AerzteAendern (arztmenge: Aerzte);

PRE-CONDITION:

(FORALL arztvar IN arztmenge:

 arztvar.ArztNr != NIL AND arztname.ArztName != NIL)

AND (* CONDITION ON ARZT *)

(FORALL arztvar IN arztmenge:

 ANY (arztold.ArztNr
 WHERE arztold.ArztNr = arztvar.ArztNr
 AND arztold.Name = arztvar.Name))

POST-CONDITION: (* CONDITION ON ARZT *)

(FORALL arztvar IN arztmenge:

 ((arzt.ALL) WHERE arzt.ArztNr = arztvar.ArztNr)

 RECMATCH arztvar;

AND (* CONDITION ON ARZT *)

((arzt.ALL) WHERE arzt.ArztNr != arztvar.ArztNr)

=

((arztold.ALL) WHERE arztold.ArztNr != arztvar.ArztNr)

(* der Vergleichsoperator RECMATCH ist dann TRUE, wenn die
zu vergleichenden Records komponentenweise übereinstim-
men, oder im zweiten Record NIL steht. *)

ACTION-END;

ACTION AerzteSuchen (arzteintrag : ArztEintrag,

 VAR arztmenge : Aerzte);

(* Der übergebene "arzteintrag" ist dabei unvollständig
belegt. Die NIL-Werte fungieren als "Wild-Cards" in der
Datenbank-Abfrage *)

Beispiel: Sicht auf die "Pharmadatenbank"

PRE-CONDITION: ...

(* in analoger Weise zu den anderen Operationen *)

MODULE-END ArztVerwaltung.

Man erkennt, daß die Spezifikation einfacher Aktionen vom gleichen Aufwand ist, wie ihre Implementierung in einer Datenbank-Manipulationssprache.

Zur Implementation des Moduls ArztVerwaltung.

DB-APPLICATION-MODULE

ArztVerwaltung (Mitarbeiter: CARDINAL);
IMPLEMENTATION;

EXPORT QUALIFIED

(* siehe Spezifikation *)

IMPORT PERSISTENT AERZTEBESTAND;

FROM AerzteGesamtVerwaltung IMPORT GetNeueArztNummern;

ACTION AerzteEintragen

(arztmenge: aerzte, VAR aerztenr: Nummernmenge);

VAR neuearztNr : CARDINAL;

VAR NeueArztNr : SET IN Nummernmenge;

BEGIN-OF-ACTION

IF EXISTS1 arztvar IN arztmenge: arztvar.ArztNr != NIL
THEN UNDO;

GetNeueArztNummer(ABS(arztmenge), NeueArztNr);

FORALL arztvar IN arztmenge DO

neuearztNr := EXTRACT NeueArztNr;

APPEND TO ARZT (ArztNr=neuearztNr

Anrede=arztvar.Anrede

:

RPM-KreisNr=arztvar.RPM-KreisNr);

END;

aerztenr := NeueArztNr;

END-OF-ACTION;

(* Die Implementation der anderen Aktionen erfolgt entsprechend *)

:

MODULE-END <appl-modul>.

Beispiel: Sicht auf die "Pharmadatenbank"

- (ii) Realisierung des Moduls ArztBesuchsVerwaltung für die Verwaltung von Arztbesuchen, Gesprächen, und Musterabgaben.

Die Schnittstelle dieses Moduls ist im Gegensatz zum vorherigen nicht auf der Basis von Mengen gestaltet, sondern von diesem Modul werden an der Schnittstelle einzelne Tupel übergeben.

Abb. A2-5.:

(* Datenbank-Anwendungsmodul ArztBesuchsVerwaltung *)

DB-Modul:

```

-----
| ArztBesuchEintragen,          |
| ArztBesuchAustragen,         |
| ArztBesuchAendern,           |
| GespraechEintragen,           |
| GespraechAustragen,          |
| GespraechAendern             |
| ...                           |
| -----                     |
|                               |
|   ARZTBESUCH MEDIKAMENTGESPRACH |
|                               |
|                               |
-----

```

DB-APPLICATION-MODULE

ArztBesuchsVerwaltung (Mitarbeiter: CARDINAL);
SPECIFICATION;

```

EXPORT QUALIFIED ArztBesuchEintragen, ArztBesuchAustragen,
                  ArztBesuchAendern,
                  GespraechEintragen, GespraechAustragen,
                  GespraechAendern, ...
                  ArztBesuch, Gespraech,
                  ...

```

```

IMPORT PERSISTENT ARZTBESUCHSBERICHTE (Mitarbeiter);
(* Sicht der Datenbank, wie in Teil 1 beschrieben *)

```

```

FROM AERZTEBESTAND (Mitarbeiter) IMPORT PERSISTENT ARZT;
FROM GEBIETE IMPORT PERSISTENT PHARMAGEBIET;
(* Aus diesen Sichten werden die abgeleiteten Relationen
   ARZT und PHARMAGEBIET für den lesenden Zugriff
   importiert *)
FROM SYSTEM IMPORT DATE;
(* Aktuelle Datumsangabe *)

```

Beispiel: Sicht auf die "Pharmadatenbank"

TYPE ArztBesuch : RECORD OF ...

:

(* Für den Recordaufbau vergleiche die Definition der Relationen der Sicht ARZTBESUCHSBERICHTE in Anhang 1 und Teil 1 zur abgeleiteten Relation *)

VAR besuchvar : ArztBesuch;

VAR gespraechsvar : Gesprach;

RANGE OF besuchhold IS ARZTBESUCH-OLD;

RANGE OF gespraechhold IS MEDIKAMENTGESPRAECH-OLD;

RANGE OF besuch IS ARZTBESUCH-NEW;

RANGE OF gespraech IS MEDIKAMENTGESPRAECH-NEW;

RANGE OF arzt IS ARZT;

ACTION ArztBesuchsEintrag (besuchvar: ArztBesuch);

(* analog zum Arzteintrag, allerdings ohne Mengenkonstrukte *)

PRE-CONDITION: (* CONDITION ON ARZT *)

ANY (arzt.ArztNr

WHERE arzt.ArztNr = besuchvar.ArztNr)

AND (* CONDITION ON ARZTBESUCH *)

NOT ANY (besuchhold.ArztNr

WHERE besuchhold.Datum = besuchvar.Datum

AND besuchhold.FolgeNr = besuchvar.FolgeNr)

AND

(besuchvar.datum <= DATE)

POST-CONDITION: (* CONDITION ON ARZTBESUCH *)

((besuch.ALL) WHERE besuch.Datum = besuchvar.Datum

AND besuch.FolgeNr = besuchvar.Folgenr)

RECMATCH besuchvar);

AND

(besuchhold.ALL)

=

((besuch.ALL) WHERE besuch.Datum != besuchhold.Datum

AND besuch.FolgeNr != besuchhold.Folgenr)

ACTION-END;

ACTION ArztBesuchAustragen (besuchvar: ArztBesuch)

PRE-CONDITION: (* CONDITION ON ARZTBESUCH *)

ANY (besuchhold.ArztNr

WHERE besuchhold.ArztNr = besuchvar.ArztNr

AND besuchhold.FolgeNr = besuchvar.FolgeNr

AND besuchhold.Datum = besuchvar.Datum)

Beispiel: Sicht auf die "Pharmadatenbank"

AND

(besuchvar.Datum <= DATE - 7 DAYS)

POST-CONDITION: (* CONDITION ON ARZTBESUCH *)

NOT ANY (besuch.ArztNr

WHERE besuch.ArztNr = besuchvar.ArztNr

AND besuch.FolgeNr = besuchvar.FolgeNr

AND besuch.Datum = besuchvar.Datum)

AND (* CONDITION ON ARZTBESUCH *)

(besuch.ALL)

=

((besuchold.ALL) WHERE besuch.ArztNr = besuchold.ArztNr

AND besuch.FolgeNr = besuchold.FolgeNr

AND besuch.Datum = besuchold.Datum

ACTION-END;

:

:

MODULE-END ArztBesuchsVerwaltung.

Die Implementation des Moduls ArztBesuchsVerwaltung erfolgt analog zum Modul ArztVerwaltung. Sie wird deshalb nicht weiter ausgeführt.

Anhang 3

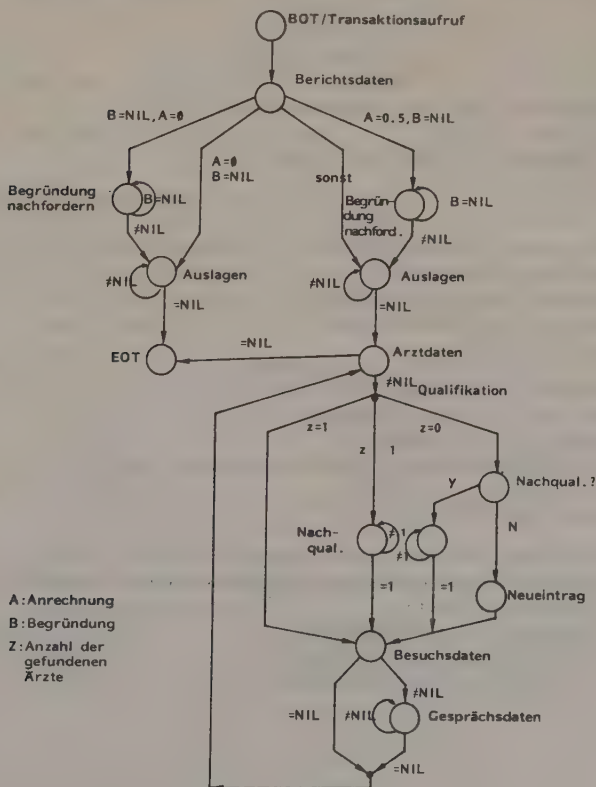
Beispiel: Einige typische Transaktionen für einen Pharmaaußendienstmitarbeiter

Zur Erläuterung des Transaktionsaufbaus werden zwei Transaktionen für den Außendienstmitarbeiter skizziert. Es werden die typischen Fälle des Erstellen eines Tagesberichts, als reglementierte Transaktionen und als freie Transaktion die Verwaltung der Ärztestammdaten

(1) Erstellen eines Tagesberichts

Abb. A3-1.

(* Dialogdiagramm vgl. Abs. 4.3.3 und <Bro82> *)



Beispiel: Transaktionen für Mitarbeiter

TRANSACTION Tagesbericht (mitarbeiter : CARDINAL);

(* Der Parameter "Mitarbeiter" wird von der
Schutzkomponente des Systems versorgt *)

SPECIFICATION;

SCOPE;

FROM DB-VIEWS IMPORT ArztVerwaltung (mitarbeiter),
ArztBesuchsVerwaltung (mitarbeiter),
DienstBerichtsVerwaltung (mitarbeiter);

FROM DB-VIEWS IMPORT MitarbeiterVerwaltung (mitarbeiter);

FROM SYSTEM IMPORT DATE;

FROM DC-SYSTEM IMPORT TagesBerichtKomModul;

VAR: berichtseintrag : Bericht;
besuchseintrag: ArztBesuch;
besuchsdaten: SET IN ArztBesuch;
gespraechseintrag: Gespraech;
gespraechsdaten: SET IN Gespraech;
auslageneintrag : Auslagen;
auslagedaten: SET IN Auslagen;
arzteintrag: Arzt;
arztdaten: SET IN Arzt;
arztmenge: SET IN Arzt;

(* ArztBesuch, etc. werden vom DC-System importiert *)

(* Tupelvariablen für den neuen Datenbankzustand *)

(* die Tupelvariablen beziehen sich auf jeweils auf
abgeleiteten mitarbeiterbezogenen Relationen (Views) *)

(* Relation aus ArztVerwaltung *)

RANGE OF arzt IS ARZT;

(* Relation aus ArztBesuchsVerwaltung *)

RANGE OF besuch IS ARZTBESUCH;

(* Relation aus MitarbeiterVerwaltung *)

RANGE OF mitarb IS MITARBEITER;

(* Relation aus DienstBerichtsVerwaltung *)

RANGE OF auslagen IS AUSLAGEN;

RANGE OF bericht IS BESUCHSBERICHT;

(* Tupelvariablen für den alten Datenbankzustand *)

(* nur zur Spezifikation der Initialbedingungen notwendig *)

RANGE OF arztold IS ARZT;

RANGE OF besuchold IS ARZTBESUCH;

RANGE OF auslagenold IS AUSLAGEN;

RANGE OF berichtold IS BESUCHSBERICHT;

Beispiel: Transaktionen für Mitarbeiter

CONDITIONS:

INIT-CONDITION:

TRIGGER FOR mitarbeiter

IN (mitarb.MA-Nr)

WHERE mitarb.Funktion = "Pharmareferent"

ON MAX (besuch.Datum + 1 WEEK) <= DATE;

PERMIT FOR mitarbeiter IN

IN (mitarb.MA-Nr)

WHERE mitarb.Funktion = "Pharmareferent"

ON berichtseintrag.Datum <= DATE AND Anrechnung = 1;

REJECT FOR Mitarbeiter

IN (mitarb.MA-Nr)

WHERE mitarb.Funktion != "Pharmareferent";

PRE-CONDITION:

ANY (mitarb.MA-Nr WHERE mitarb.MA-Nr = mitarbeiter)

AND NOT ANY (bericht.Datum

WHERE berichtold.Datum = berichtseintrag.Datum)

AND NOT ANY (auslagen.Datum

WHERE auslagenold.Datum=berichtseintrag.Datum)

AND NOT ANY (besuch.Datum

WHERE besuch.Datum = berichtseintrag.Datum)

AND NOT ANY (gespräch.Datum

WHERE gespräch.Datum = berichtseintrag.Datum)

POST-CONDITION:

(* CONDITION (1) *)

(berichtsdaten.Anrechnung = 0.0 AND auslagedaten != EMPTY

--->

((bericht.ALL)

WHERE bericht.Datum = berichtseintrag.Datum)

= berichtseintrag

AND ((auslagen.ALL)

WHERE auslagen.Datum = berichtseintrag.Datum)

= auslagedaten

(* alle anderen Einträge bleiben unverändert *)

AND ((auslagen.ALL)

WHERE auslagen.Datum != berichtseintrag.Datum)

= (auslagenold.ALL)

AND ((bericht.ALL)

WHERE bericht.Datum != berichtseintrag.Datum)

= (berichtold.ALL)

AND (besuch.ALL) = (besuchold.ALL)

AND (arzt.ALL) = (arztold.ALL)

AND (gespräch.ALL) = (gesprächold.ALL)

Beispiel: Transaktionen für Mitarbeiter

OR

(* CONDITION (2) *)

(berichtseintrag.Anrechnung = 0.0 AND auslagedaten = EMPTY

---->

((bericht.ALL)

WHERE bericht.Datum = berichtseintrag.Datum)

, = berichtseintrag

(* alle anderen Einträge bleiben unverändert *)

AND ((bericht.ALL)

WHERE bericht.Datum != berichtseintrag.Datum)

= (berichtold.ALL)

AND (auslagen.ALL) = (auslagenold.ALL)

AND (besuch.ALL) = (besuchold.ALL)

AND (arzt.ALL) = (arztold.ALL)

AND (gespräch.ALL) = (gesprächold.ALL)

OR

(* CONDITION (3) *)

berichtseintrag.Anrechnung != 0.0 AND besuchsdaten != EMPTY

AND gespraechsdaten != EMPTY

AND FORALL besuchseintrag IN besuchsdaten:

ANY (arztold.ArztNr

WHERE arztold.ArztNr = besuchseintrag.ArztNr)

---->

((bericht.ALL)

WHERE bericht.Datum = berichtseintrag.Datum)

= berichtseintrag

AND ((auslagen.ALL)

WHERE auslagen.Datum = berichtseintrag.Datum)

= auslagedaten

AND ((besuch.ALL)

WHERE besuch.Datum = berichtseintrag.Datum)

= besuchsdaten

AND ((gespraech.ALL)

WHERE gespraech.Datum = berichtseintrag.Datum)

= gespraechsdaten

(* durch Integritätsbedingungen wird gewährleistet,

daß zu jedem Gesprächseintrag auch ein Besuchseintrag
existiert. *)

AND ...

OR

(* CONDITION (4) *)

(bericht.Anrechnung != 0.0 AND besuchsdaten != EMPTY

AND gesprächsdaten = EMPTY

---->

((bericht.ALL)

WHERE bericht.Datum = berichtseintrag.Datum)

= berichtseintrag

```
AND ((auslagen.ALL)
      WHERE auslagen.Datum = berichtseintrag.Datum)
      = auslagedaten
AND ((besuch.ALL)
      WHERE besuch.Datum = berichtseintrag.Datum)
      = besuchsdaten
AND ...

OR
(* CONDITION (5) *)
(Bericht.Anrechnung != 0.0 AND besuchsdaten = EMPTY ...
--->      ...

OR
(* CONDITION (6) *)
(* Notwendigkeit zum Eintrag neuer Ärzte *)
berichtseintrag.Anrechnung != 0.0
AND besuchsdaten != EMPTY AND gespraechsdaten != EMPTY
AND NOT FORALL besuchseintrag IN besuchsdaten:
      ANY (arztold.ArztNr
            WHERE besuchseintrag.ArztNr = arztold.ArztNr)
--->
      ((bericht.ALL)
        WHERE bericht.Datum = berichtseintrag.Datum)
        = berichtseintrag
AND ((auslagen.ALL)
      WHERE auslagen.Datum = berichtseintrag.Datum)
      = auslagedaten
AND ((arzt.ALL)
      WHERE (NOT ANY (arztold.ArztNr
                      WHERE arzt.ArztNr = arztold.ArztNr))
            = arztold.ArztNr)
      = arztold.ArztNr
AND ((besuch.ALL)
      WHERE besuch.Datum = berichtseintrag.Datum)
      = besuchsdaten
AND ((gespraech.ALL)
      WHERE gespraech.Datum = berichtseintrag.Datum)
      = gespraechsdaten
AND ...
```

Die Entscheidung, z.B. ob neue Ärzte einzutragen sind, fällt während des Dialogs. Apriori zu Beginn der Transaktion ist dies nicht entscheidbar. Die vorliegende Situation entspricht illustriert die in Abs. 4.3.2 gemachten Aussagen. Die Nachbedingung ist ansonsten identisch zu der Bedingung ohne den Arzteintrag. Über die Ingeritätsbedingung wird die Korrektheit der Ärzteeinträge abgesichert (vgl. Anhang 1).

Beispiel: Transaktionen für Mitarbeiter

Zur Implementation der Transaktion "Erstellen eines Tagesbericht" wird entsprechend dem Dialogdiagramm (Abb. A3-1) vorgegangen. Die Implementation wird nur in einigen Auszügen angegeben.

```
TRANSACTION Tagesbericht (mitarbeiter : CARDINAL);
IMPLEMENTATION;

SCOPE;
FROM DB-VIEWS IMPORT ArztVerwaltung (mitarbeiter),
                    ArztBesuchsVerwaltung (mitarbeiter),
                    DienstBerichte (mitarbeiter);

FROM DB-VIEWS IMPORT MitarbeiterVerwaltung (mitarbeiter);

FROM SYSTEM IMPORT DATE;
FROM DC-SYSTEM IMPORT TagesBerichtKom;

VAR: berichtseintrag : Bericht;
    besuchseintrag : ArztBesuch;
    :
    (* vgl. Spezifikation *)

(* aus Aerzte *)
RANGE OF arzt IS ARZT;
(* aus ArztBesuche *)
RANGE OF besuch IS ARZTBESUCH;
(* aus MitarbeiterVerwaltung *)
RANGE OF mitarb IS MITARBEITER;
(* aus DienstBerichte *)
RANGE OF auslagen IS AUSLAGEN;
RANGE OF bericht IS BESUCHSBERICHT;

BEGIN-OF-TRANSACTION;
IF RETRIEVE ( NOT ANY (mitarb)
              WHERE mitarb.MA-Nr = Mitarbeiter )
    THEN REJECT;

SAVEPOINT:
REQUEST (berichtseintrag) COND BACKUP;
IF NOT berichtseintrag.Datum <= DATE
    AND berichtseintrag.Anrechnung = 1
    AND ((mitarb.Funktion) WHERE mitarb.MA-Nr = Mitarbeiter)
        = "Pharmareferent"
    THEN ...
    (* Aktionen zur Information des Vorgesetzten *)

IF berichtseintrag.Anrechnung = 0.0
THEN WHILE berichtsseintrag.Begrundung = NIL
    DO REQUEST (berichtseintrag.Begründung) COND BACKUP;
    END;
```


Beispiel: Transaktionen für Mitarbeiter

```
BerichtEintragen (berichtseintrag);
auslageneintrag = NIL;
auslageneintrag.BelegNr = 0;
WHILE auslageneintrag.Kostenart != NIL
DO auslageneintrag.BelegNr = auslagen.BelegNr + 1;
   auslageneintrag.Datum = berichtseintrag.Datum;
   REQUEST (auslageneintrag) COND BACKUP;
   IF auslageneintrag != NIL
       THEN AuslagenEintragen (auslageneintrag);
           auslagedaten += auslageneintrag
       END
   END
END;
```

```
ELSE WHILE berichtseintrag.Begrundung = NIL
      AND berichtseintrag.Anrechnung = 0.5
DO REQUEST (besuchseintrag.Begründung) COND BACKUP;
END;
BerichtEintragen (berichtseintrag);
auslageneintrag = NIL;
auslageneintrag.BelegNr = 0;
WHILE auslageneintrag.Kostenart != NIL
DO auslageneintrag.BelegNr = auslagen.BelegNr + 1;
   auslageneintrag.Datum = berichtseintrag.Datum;
   REQUEST (auslageneintrag) COND BACKUP;
   IF auslageneintrag != NIL
       THEN AuslagenEintragen (auslageneintrag);
           auslagedaten += auslageneintrag
       END
   END
END;
```

```
SAVEPOINT:
WHILE REQUEST (arzteintrag) != NIL COND BACKUP;
DO ArztSuchen (arzteintrag, arztmenge)
   IF ABS(arztmenge) = 0
       THEN DISPLAY ("nicht gefunden");
           SAVEPOINT:
           REQUEST ("Nachqualifikation") COND BACKUP;
           IF Nachqualifikation
               THEN WHILE ABS(arztmenge) != 1
                   DO REQUEST (arzteintrag) != NIL
                       COND BACKUP;
                   ArztSuchen (arzteintrag, arztmenge);
               END;
           ELSE REQUEST (arzteintrag) COND BACKUP;
               AetzteEintragen (arzteintrag);
               arztdaten += arzteintrag
           END;
END;
```

Beispiel: Transaktionen für Mitarbeiter

```
ELSE IF ABS(arztmenge) > 1
  THEN WHILE ABS(arztmenge) != 1
    DO REQUEST (arzteintrag) != NIL
      COND BACKUP;
      ArztSuchen (arzteintrag, arztmenge);
    END;
  END;
END;

SAVEPOINT:
IF REQUEST (besuchseintrag) != NIL COND BACKUP;
THEN  ArztBesuchEintragen (besuchseintrag);
      besuchsdaten += besuchseintrag;
      WHILE REQUEST (gespraechseintrag) != NIL
        COND BACKUP;
        DO MedikamentGespraechEintragen
          (gespraechseintrag);
          gespraechsdaten += gespraechseintrag;
        END
      END
END
END;
IF NOT .... THEN UNDO
      ELSE END-OF-TRANSACTION
```

Eine Prüfung der Post-Condition kann entfallen. Die strukturellen und semantischen Bedingungen können bereits auf der Basis der Aktionen berücksichtigt werden. Es verbleibt die Prüfung bestimmter finaler deontischer Bedingungen, z.B. ob der Pharmareferent weniger als 8 Besuche gemacht hat und als Anrechnungsfaktor 1 angegeben wurde, in diesem Fall wäre eine Benachrichtigung des Gruppenleiters u. U. sinnvoll.

Beispiel: Transaktionen für Mitarbeiter

(2) Verwalten der Ärztestammdaten

Zur Erläuterung einer freien Transaktion die wird Verwaltung der Ärztestammdaten angegeben. Die Verwaltung (Ändern, Löschen, Eintragen, Listen) kann auf der Basis des im Anhang 2 spezifizierten Anwendungsmoduls ArztVerwaltung vorgenommen werden. Die Mengenoperationen sind lediglich aufgrund der Beschränkung der peripheren Geräte (Terminal) in der Erstellung der zu ändernden oder eintragenden Mengen von Ärzten für den Benutzer in eine sequentielle Abarbeitung zu bringen.

```
TRANSACTION AertzteStammdatenPflege (Mitarbeiter);
SPECIFICATION;
SCOPE;
```

```
FROM DB-VIEWS IMPORT Aerzte (Mitarbeiter),
```

```
FROM DB-VIEWS IMPORT MitarbeiterVerwaltung;
```

```
FROM SYSTEM IMPORT DATE;
```

```
FROM DC-SYSTEM IMPORT AertzteKomModul;
```

```
(* analog zur Transaktion Tagesbericht *)
```

```
(* aus MitarbeiterVerwaltung *)
```

```
RANGE OF mitarb IS MITARBEITER;
```

```
RANGE OF mitarb2 IS MITARBEITER;
```

```
RANGE OF gebiet IS PHARMAGEBIET;
```

```
:
```

```
CONDITIONS:
```

```
INIT-CONDITION:
```

```
PERMIT FOR Mitarbeiter IN (mitarb.MA-Nr)
```

```
    WHERE mitarb.Funktion = "Pharmareferent"
```

```
    AND Mitarbeiter = mitarb.MA-Nr
```

```
PERMIT FOR Mitarbeiter
```

```
    IN RETRIEVE (mitarb.MA-Nr)
```

```
    WHERE mitarb.Funktion = "Gebietsleiter"
```

```
    AND mitarb.MA-KreisNr = gebiet.OrgEinheit
```

```
    AND gebiet.PhGeNr = mitarb2.MA-KreisNr
```

```
    AND mitarb2.MA-Nr = Mitarbeiter
```

```
REJECT FOR Mitarbeiter
```

```
    NOT IN ....
```

```
PRE-CONDITION: TRUE
```

```
POST-CONDITION: TRUE
```

```
END AerzteVerwaltung.
```

```
TRANSACTION AerzteStammdatenPflege (Mitarbeiter);
IMPLEMENTATION;
SCOPE;

FROM DB-VIEWS IMPORT ArztVerwaltung (Mitarbeiter);

FROM DB-VIEWS IMPORT MitarbeiterVerwaltung;

FROM SYSTEM IMPORT DATE;
FROM DC-SYSTEM IMPORT TagesBerichtKom;

(* aus Aerzte *)
RANGE OF arzt IS ARZT;
(* aus MitarbeiterVerwaltung *)
RANGE OF mitarb IS MITARBEITER;

BEGIN-OF-TRANSACTION;

IF NOT (ANY (mitarb.MA-Nr) WHERE mitarb.MA-Nr = Mitarbeiter)
  THEN REJECT;
  IF NOT (RETRIEVE (mitarb.Funktion)
    WHERE mitarb.MA-Nr = Mitarbeiter
      = "Pharmareferent"
    OR
  ..
    THEN ...
    (* Aktionen zur Information des Vorgesetzten *)

SAVEPOINT:
WHILE REQUEST(Funktion) != NIL
DO CASE Funktion OF
  Aendern:
    neuqual := TRUE
    WHILE neuqual
    DO SAVEPOINT:
      REQUEST (arztsuchbedingung) COND BACKUP;
      AerzteSuchen (arztsuchbedingung, arztmenge);
      DISPLAY (arztmenge);
      REQUEST (neuqual) COND BACKUP;
    END;
    REQUEST (arztmenge) COND BACKUP;
    AerzteAendern (arztsuchbedingung, arztmenge);
1: Loeschen:
    neuqual := TRUE
    WHILE neuqual
    DO SAVEPOINT:
      REQUEST (arztsuchbedingung) COND BACKUP;
      AerzteSuchen (arztsuchbedingung, arztmenge);
      DISPLAY (arztmenge);
      REQUEST (neuqual) COND BACKUP;
    END;
```

Beispiel: Transaktionen für Mitarbeiter

```
    REQUEST (arztmenge) COND BACKUP;
    AerzteAustragen (arztmenge);
! : Eintragen:
    neuqual := TRUE
    WHILE neuqual
    DO SAVEPOINT:
        REQUEST (arztsuchbedingung) COND BACKUP;
        AerzteSuchen (arztsuchbedingung, arztmenge);
        DISPLAY (arztmenge);
        REQUEST (neuqual) COND BACKUP;
    END;
    REQUEST (arztmenge) COND BACKUP;
    AerzteEintragen (arztmenge, arztmr);
! : Listen:
    neuqual := TRUE
    WHILE neuqual
    DO SAVEPOINT:
        REQUEST (arztsuchbedingung) COND BACKUP;
        AerzteSuchen (arztsuchbedingung, arztmenge);
        DISPLAY (arztmenge);
        REQUEST (neuqual) COND BACKUP;
    PRINT (arztmenge);
END;
END-OF-TRANSACTION;
```

Anhang 4

Beispiel: Einige typische Transaktionen
für die Pharmaaußendienstleitung

Als Beispiel für eine schematische Transaktion wird der statistische Monatsbericht über die Arztbesuche der Pharmareferenten angegeben, den die Leitung des Pharmaaußendienstes zu Kontrollzwecken benötigt.

Diese schematische Transaktion ist sehr einfach. Sie wird am Ende eines Monats von Datenbanksystem initiiert und erzeugt eine Liste aller Pharmareferenten mit der Ausgabe ihrer Besuchsschnitte.

TRANSACTION Besuchsschnitt.
SPECIFICATION;
SCOPE;

FROM SYSTEM IMPORT DATE;

TYPE AUSGABE = RELATION OF RECORD
MA-Nr : CARDINAL;
Name : NAME;
PhGeNr : CARDINAL;
Arbeitstage : 0 .. 31 OF CARDINAL;
Besuche : CARDINAL;
Besuchsschnitt: REAL;
Anteil-A-Aerzte : 0 .. 100 OF CARDINAL;
Anteil-B-Aerzte : 0 .. 100 OF CARDINAL;
Anteil-C-Aerzte : 0 .. 100 OF CARDINAL;
Anteil-N-Aerzte : 0 .. 100 OF CARDINAL;

RANGE OF ausg IS AUSGABE;
RANGE OF arzt IS ARZT;
RANGE OF besuch IS ARZTBESUCH;
RANGE OF bericht IS BESUCHSBERICHT;
RANGE OF mitarb IS MITARBEITER;

CONDITIONS;

INIT-CONDITION:

TRIGGER FOR SYSTEM ON DATE.Day = 1 AND TIME = 7.00.00

PRE-CONDITION: TRUE

Beispiel: Transaktionen für Außendienstleitung

POST-CONDITION: ...

(* Die Transaktion verändert keine Daten der Datenbank.
Das Prädikat zur Beschreibung der Ausgabe entspricht
genau der Datenbankabfrage der Implementation *)

TRANSACTION Besuchsschnitt.

IMPLEMENTATION;

SCOPE;

FROM SYSTEM IMPORT DATE;

TYPE AUSGABE = RELATION OF RECORD

MA-Nr : CARDINAL;

Name : NAME;

PhGeNr : CARDINAL;

Arbeitstage : 0 .. 31 OF CARDINAL;

Besuche : CARDINAL;

Besuchsschnitt: REAL;

Anteil-A-Aerzte : 0 .. 100 OF CARDINAL;

Anteil-B-Aerzte : 0 .. 100 OF CARDINAL;

Anteil-C-Aerzte : 0 .. 100 OF CARDINAL;

Anteil-N-Aerzte : 0 .. 100 OF CARDINAL;

RANGE OF ausg IS AUSGABE;

RANGE OF arzt IS ARZT;

RANGE OF besuch IS ARZTBESUCH;

RANGE OF bericht IS BESUCHSBERICHT;

RANGE OF mitarb IS MITARBEITER;

BEGIN-OF-TRANSACTION;

CREATE AUSGABE (....

RETRIEVE INTO AUSGABE

(ausg.MA-Nr = mitarb.MA-Nr,

ausg.Name = mitarb.Name,

ausg.PhGeNr = mitarb.PhGeNr,

ausg.Arbeitstage =

SUM(bericht.Anrechnung

WHERE mitarb.MA-Nr = bericht.MA-Nr

AND bericht.Datum.Month = DATE.Month-1)

ausg.Besuche =

COUNT(besuch.ArztNr

WHERE mitarb.MA-Nr = besuch.MA-Nr

AND besuch.Datum.Month = DATE.Month-1)

ausg.Besuchsschnitt =

COUNT(besuch.ArztNr

WHERE mitarb.MA-Nr = besuch.MA-Nr

AND besuch.Datum.Month = DATE.Month-1)

/ SUM(bericht.Anrechnung

WHERE mitarb.MA-Nr = bericht.MA-Nr

AND bericht.Datum.Month = DATE.Month-1)

Beispiel: Transaktionen für Außendienstleitung

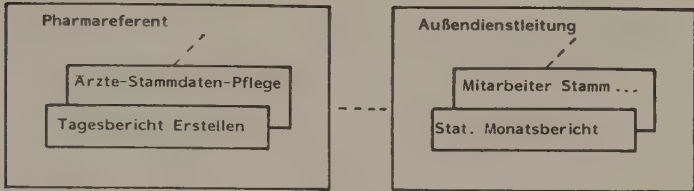
```
      ausg.Anteil-A-Aerzte =
        COUNT(besuch.ArztNr
          WHERE mitarb.MA-Nr = besuch.MA-Nr
            AND besuch.Datum.Month = DATE.Month-1)
            AND besuch.ArztNr = arzt.ArztNr
            AND arzt.Kategorie = "A"
        / COUNT(besuch.ArztNr
          WHERE mitarb.MA-Nr = besuch.MA-Nr
            AND besuch.Datum.Month = DATE.Month-1)
      * 100
      ausg.Anteil-B-Aerzte =
        COUNT(besuch.ArztNr
          WHERE mitarb.MA-Nr = besuch.MA-Nr
            AND besuch.Datum.Month = DATE.Month-1)
            AND besuch.ArztNr = arzt.ArztNr
            AND arzt.Kategorie = "B"
        / COUNT(besuch.ArztNr
          WHERE mitarb.MA-Nr = besuch.MA-Nr
            AND besuch.Datum.Month = DATE.Month-1)
      * 100
      ausg.Anteil-C-Aerzte =
        COUNT(besuch.ArztNr
          :
          : )
        WHERE mitarb.Funktion = "Pharmareferent"
        SORT BY ausg.Besuchsschnitt
PRINT (AUSGABE)
DESTROY AUSGABE
END-OF-TRANSACTION.
```

Es wurde in diesem Fall zur Implementierung direkt auf die Datenbank-Abfragesprache zurückgegriffen. Die RETRIEVE-Anweisung könnte man z.B. auch als eine Aktion in einen Modul "StatistischeBerichte" realisieren. Die schematische Transaktion reduziert sich dann auf die Triggerung, d.h. den Aufruf der Aktion und die Ausgabe des Ergebnisses.

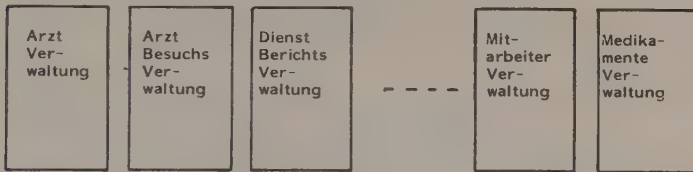
Anhang 5

Überblick: Ausschnitt aus dem Datenbank-Anwendungssystem
 für den Pharmaaußendienst

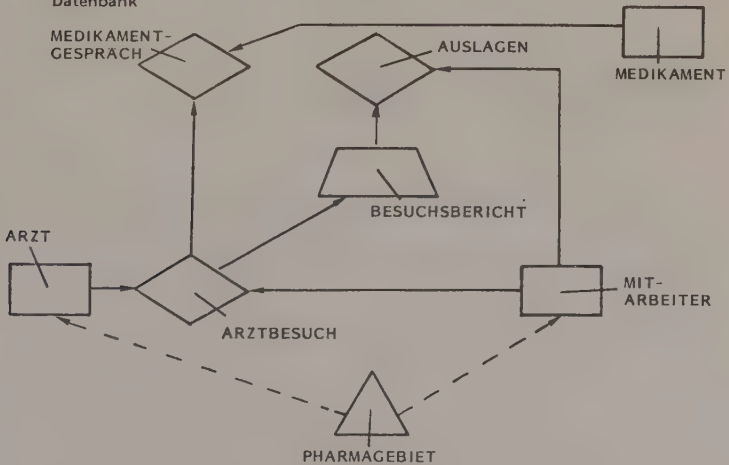
Anwendung:



Anwendungsmodule



Datenbank



LEBENS LAUF

Dieter Steinbauer
Diplom-Informatiker

geb. am 25.11.1950
in Gunzenhausen

verheiratet

1957 - 1962	Besuch der Volksschule in Gunzenhausen
1962 - 1972	Besuch des Simon-Marius-Gymnasiums in Gunzenhausen (mathe.-naturw. Zweig)
23.6.1972	Abitur
Okt. 1972 bis Sept. 1978	Studium der Informatik an der Friederich-Alexander-Universität Erlangen-Nürnberg
5.10.1978	Diplom
1.9.1978 bis 31.3.1980	Mitarbeiter der Firma SIEMENS AG in Erlangen im Bereich Steuerungs- und Automatisierungs- technik
seit 1.4.1980	Wissenschaftlicher Mitarbeiter am IMMD (Informatik) VI Lehrstuhl für Datenbanksysteme

In letzter Zeit sind folgende Arbeitsberichte des Instituts für
Mathematische Maschinen und Datenverarbeitung erschienen:
=====

Band 13

=====

- Nr. 1 Schrammel, D., Bayer, W., Gliemann, H.-P., Heigl, M.,
Mertens, P.:
Ein Leistungs- und Wirtschaftlichkeitsvergleich zwischen
Klein- und Großcomputern
(Januar 1980)
- Nr. 2 Göttler, H.:
Semantische Modelle
(März 1980)
- Nr. 3 Fromm, H.J.:
Zur Modellierung der Speicherinterferenz bei hierarchisch
organisierten Multiprozessorsystemen
(April 1980)
- Nr. 4 Jäpel, D.:
Klassifikatorbezogene Merkmalsauswahl
(Juli 1980)
- Nr. 5 Bodendorf, F., Mertens, P., Haas, R., Weber, G., Wolf, G.:
SAMBA - ein Methodenbankrahmen um das Statistik-Paket SPSS
(Juli 1980)
- Nr. 6 Bartsch, B.:
Inferenz und Analyse spezieller Graphgrammatiken für die
syntaktische Mustererkennung
(September 1980)
- Nr. 7 Wurm, F.X.:
Auftragssystem für eine Mehrprozessoranlage
(Oktober 1980)
- Nr. 8 Schedelbeck, A.:
Vergleich der beiden höheren Realzeitprogrammiersprachen
ADA und PEARL
(Oktober 1980)
- Nr. 9 Jahresbericht 1980 der Informatik
(März 1981)

- Nr. 1 Pattern Recognition
 (Research at Lehrstuhl für Informatik 5)
 (Februar 1981)
- Nr. 2 Bunke, H.:
 Analyse elektrischer Schaltpläne
 (März 1981)
- Nr. 3 Linster, C.-U.:
 SYMPOS/UNIX - Ein Betriebssystem für homogene
 Polyprozessorsysteme
 (Juni 1981)
- Nr. 4 Akyildiz, J.F., Bolch, G.:
 Analytic Solution Techniques for Queueing Network
 Models of Computer Systems
 (Juli 1981)
- Nr. 5 Lindstedt, W.G.:
 Ein Verfahren zur Erstellung portabler algorithmischer
 Laufzeitprogramme
 (Juli 1981)
- Nr. 6 Vollmar, R.:
 Zum Begriff des Parallelismus bei Polyautomaten
 (Juli 1981)
- Nr. 7 Luft, A.L.:
 Rationaler Sprachgebrauch und orthosprachliche
 Standardisierung als Grundlagen zuverlässiger Software-
 Entwicklung und -Dokumentation
 (Januar 1982)
- Nr. 8 Händler, W.:
 PROCEEDINGS OF A WORKSHOP ON TAXONOMY IN COMPUTER
 ARCHITECTURE
 (Februar 1982)
- Nr. 9 Beth, Th., Hess, P., Wirl, K.:
 Materialien zur Kryptographie
 (März 1982)
- Nr. 10 Jahresbericht 1981 der Informatik
 (März 1982)

- Nr. 1 Wittmann, A.:
Ein Mechanismus für die Synchronisation paralleler
Prozesse
(Februar 1982)
- Nr. 2 Maehle, E.:
Fehlertolerantes Verhalten in Multiprozessoren -
Untersuchungen zur Diagnose und Rekonfiguration
(März 1982)
- Nr. 3 Bathelt, P.:
Vergleich von Synchronisationsmechanismen
(Mai 1982)
- Nr. 4 Keramidis, S.:
Eine Methode zur Spezifikation und korrekten
Implementierung von asynchronen Systemen
(Juni 1982)
- Nr. 5 Fromm, H.J.:
Multiprozessor-Rechenanlagen:
Programmstrukturen, Maschinenstrukturen und Zuordnungs-
probleme
(Mai 1982)
- Nr. 6 Bley, H.:
Vorverarbeitung und Segmentierung von Stromlaufplänen
unter Verwendung von Bildgraphen
(Juli 1982)
- Nr. 7 Hohl, W.:
Informatik-Sammlung
Katalog mit historischen Erläuterungen
(August 1982)
- Nr. 8 Messerer, M.:
Ein neuer Ansatz zur Parallelisierung von
Compilern
(August 1982)
- Nr. 9 Grosch, J.:
Eine Programmiersprache mit mengentheoretischen
Konstrukten und deren effiziente Implementierung
(August 1982)
- Nr. 10 Kleinöder, W.:
Stochastische Bewertung von Aufgabenstrukturen
für hierarchische Mehrrechnersysteme
(August 1982)

- Nr. 11 Kneißl, F.:
Realisierung von Makro-Datenflußmechanismen auf
hierarchischen Mehrrechnersystemen
(August 1982)
- Nr. 12 Brendel, W., Fan, Z., Klar, R., Schmielau, W.:
ERES 82
Handbuch und Fallstudie
Handbook and Case Study
(Oktober 1982)
- Nr. 13 Pelz, K.:
Portable Codegenerierung in Übersetzern
für Prozeß-Programmiersprachen
(Oktober 1982)
- Nr. 14 Sommer, U., Steinchen, J.:
Ein Kleinrechner-Grafiksystem zur Dokumentation
von EDV-Abläufen
(Oktober 1982)
- Nr. 15 Hein, H.-W.:
Das Erlanger Spracherkennungssystem
(November 1982)
- Nr. 16 Bolch, G., Bruchner, W.:
Analytische Modelle für symmetrische Mehrprozessor-
anlagen mit dynamischen Prioritäten
(Dezember 1982)
- Nr. 17 Jahresbericht 1982 der Informatik
(März 1983)

- Nr. 1 Gall, R.:
Formale Beschreibung des inkrementellen Programmierens-
im-Großen mit Graph-Grammatiken
(März 1983)
- Nr. 2 König, R.:
Beiträge zur algebraischen Theorie der formalen Sprachen
(März 1983)
- Nr. 3 Welbourn, D.B.:
The Design of Mechanical Components and the Development
of DUCT:
17 Years of CAD/CAM in Cambridge University
(April 1983)
- Nr. 4 Woitok, R.:
Die System-Implementierungs-Sprache ALICE
(Mai 1983)
- Nr. 5 Schneider, S., Schwab, P., Renzinger, W.:
Wesen, Vergleich und Stand von Software zur Produktion
von Systemen der computergestützten Unternehmensplanung
(Juli 1983)
- Nr. 6 Händler, W., Herzog, U., Hofmann, F., Schneider, H.J.
und Mitarbeiter:
Projektabschlußbericht 1978 - 1982 Erlangen General Purpose
Array
(Juli 1983)
- Nr. 7 Mackert, L.:
Modellierung, Spezifikation und korrekte Realisierung
von asynchronen Systemen
(Juli 1983)
- Nr. 8 Reitenspieß, M.:
Sprachkonstrukte zur Spezifikation und korrekten Implemen-
tierung von Schutzproblemen
(Juli 1983)
- Nr. 9 Weber, K.:
Modellierung von Fehlverhalten mit Berücksichtigung
paralleler Abläufe
(August 1983)
- Nr. 10 Dietsch, H.:
Ein verteiltes Kommunikationssystem für einen Verbund
aus lokalen Mikrorechner-Netzwerken
(Dezember 1983)

- Nr. 11 Maehle, E., Schmitter, E.:
Workshop Fehlertolerante Mehrprozessor- und Mehr-
rechnersysteme
(Dezember 1983)
- Nr. 12 Volkert, J.:
Multiplikative Abstandsfunktionen und deren Anwendung
in der Bildverarbeitung
(Dezember 1983)

